University of Alberta

A Heuristic-Based Approach to Multi-Agent Moving-Target Search

by

Alejandro Isaza

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2008

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-47268-2
Our file Notre référence
ISBN: 978-0-494-47268-2

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

The main focus of this study is to consider strategies for several agents that try to capture a single moving target, that is, multi-agent moving-target search. Until now, researchers have treated the moving-target search problem as a single-agent search problem by using single-agent search algorithms such as A* and single-agent search heuristic functions such as Manhattan distance. This thesis shows how coordinating multiple agents can be achieved with a simple algorithm, CRA, that uses the *cover* heuristic — a new heuristic function designed for multi-agent search problems. CRA improves the state-of-the-art on single- and multi-agent moving-target search. We show that CRA is optimal in graphs for which one pursuer has a winning strategy and we compare the performance of CRA with *optimal* strategies on small problems, showing that CRA is within 10% of optimal on this set of problems. Finally, we show that CRA has a higher success rate and a lower capture time than the previous state-of-the-art algorithms.

Table of Contents

Introduction 1

1 Related Work 4
- 1.1 Theoretical properties of pursuit on a graph . . . 4
- 1.2 Single-Agent Search . . . 6
- 1.3 Moving-Target Search . . . 6
- 1.4 State-space abstraction . . . 9
- 1.5 Other approaches . . . 10

2 Theoretical Properties 12
- 2.1 Uniform speed . . . 12
- 2.2 Non-uniform speed . . . 16

3 Heuristic Methods 20
- 3.1 The geometric distance heuristic . . . 20
- 3.2 The cover heuristic . . . 22
  - 3.2.1 Non-uniform speed . . . 23
  - 3.2.2 Graphs . . . 25
  - 3.2.3 Advantages and disadvantages . . . 25
- 3.3 CRA . . . 27
  - 3.3.1 Risk . . . 28
  - 3.3.2 Abstraction . . . 28
  - 3.3.3 Tie-breaking . . . 29
  - 3.3.4 Avoiding local minima . . . 29
  - 3.3.5 Putting it together: CRA . . . 29
- 3.4 Target's Cover . . . 30
- 3.5 Theoretical properties of Cover . . . 31

4 Empirical Evaluation 33
- 4.1 Optimal solution . . . 34
- 4.2 Coordination . . . 37
- 4.3 Abstraction . . . 38
- 4.4 Risk . . . 41

Conclusions 44

Bibliography 47

List of Tables

4.1 The pursuit algorithms. 33
4.2 The target algorithms. 34
4.3 The optimal time for the subset of the problems solved by each algorithm and the time the algorithm took to solve them. 37

List of Figures

1.1 A planar graph in which 3 pursuers are necessary to capture a target. 5
1.2 LRTA* algorithm 7
1.3 MTS algorithm 7
1.4 The second version of the MTS algorithm 8

2.1 The algorithm to calculate the optimal strategies and to determine the capturability of the target for all starting configurations. 13
2.2 A grid-world in which three pursuers are necessary. 16
2.3 A grid map in which a target can escape from a pursuer. 18

3.1 A map in which a geometric distance heuristic does a bad job of capturing a target. 21
3.2 The set of points to which both agents can get to at the same time separates the pursuer's territory from the target's territory. 22
3.3 The cover set when the pursuer is slower than the target. 25
3.4 Cover set calculation algorithm. 26
3.5 The cover set of two pursuers and one target with the same speed. The second pursuer's cover is inside the first pursuer's cover. 27
3.6 A local minimum of the cover heuristic. 27
3.7 The CRA algorithm. 30
3.8 A graph in which one pursuer can always capture one target. By removing pitfalls it can be reduced to a single vertex. 32

4.1 The success rate on five different maps of several pursuer algorithms after several target algorithms. 35
4.2 A hand-made map with obstacles 38
4.3 The success rate as a function of the number of pursuers on an empty 20×20 grid for different speeds. 39
4.4 The success rate as a function of the number of pursuers on a grid-world with obstacles for different speeds. 39
4.5 A map from a commercial video game. 40
4.6 The run time per move, per agent for the pursuit algorithms. For CRA the time is shown for different levels of abstraction. 40
4.7 The success rate of the pursuit algorithms using two pursuers and chasing TGC, DAM and SFL. For CRA the success rate is shown for different levels of abstraction. 42
4.8 The success rate of the pursuit algorithms using three pursuers and chasing TGC, DAM and SFL. For CRA the success rate is shown for different levels of abstraction. 42
4.9 The success rate of the pursuit algorithms when chasing TGC, DAM and SFL. For CRA the success rate is shown for different values of risk. 43

Introduction

One of the central problems in artificial intelligence (AI) is single-agent search. Often, the objective of intelligent agents is to reach a specific state, or a goal, while minimizing the cost. This happens in a graph $G = (V, E)$, where V is the set of vertices (states) and E is the set of edges (actions). One way to achieve the objective is to consider all possible sequences of actions and decide which one, if any, reaches the goal with the minimum cost.

Very little attention has been given to a closely related problem: moving-target search (also known as moving-target pursuit). In this problem the goal is no longer to reach a fixed state but rather to reach the state occupied by another agent, the target. The problem then becomes a two-player game: the optimal solution for the pursuer depends on what the target does and the optimal solution for the target depends on what the pursuer does. The game can be turn based: first agent a moves to a neighboring vertex or stays in its current vertex, then agent b moves (or stays) to complete a *round*. The game starts at round 0 with agent a at vertex $v_a(0)$ and agent b at vertex $v_b(0)$. At each round, $t > 0$, agent a moves from $v_a(t-1)$ to $v_a(t)$ and then agent b moves from $v_b(t-1)$ to $v_b(t)$. The game ends with a capture if agent a and agent b are in the same vertex: $v_a(t_c) = v_b(t_c)$ or $v_a(t_c) = v_b(t_c - 1)$. The round at which this occurs, t_c, is the *capture time*. If after t_{timeout} steps there is no capture the game ends with a timeout.

The agents can be both cooperating to reach each other, or perhaps one of them may be evading the other, or even one of them may be oblivious of the other. Each case may require a different *strategy*. A strategy for agent a is a function $\sigma_a : V \times V \to V$ that, for each possible vertex for a and b, gives the vertex $v_a(t+1) = \sigma_a(v_a(t), v_b(t))$ that should be chosen next for agent a. An *optimal strategy* for the pursuers is one that minimizes the capture time. A *winning strategy* is a strategy which always captures the target, in the case of the pursuers, or always escapes, in the case of the target, regardless of the opponent's strategy.

Being two-player games, these problems can be solved doing a minimax search in the joint state-space. Unfortunately the time complexity of this approach grows exponentially with the number of agents and therefore is not viable for most practical applications. Actually, determining the number of pursuers necessary to capture a target in arbitrary graphs is EXPTIME-complete [14], and determining if n pursuers can capture a target takes time $\Omega(|V|^{2(n+1)})$.

Previous approaches address the single-agent moving-target search problem by using single-

agent search techniques. One such approach is MTS [22], which conducts a real-time search whose goal is the current location of the target, usually using Manhattan distance as heuristic function. As will be seen later, there are other approaches that are better-suited for the moving-target problem.

An even more interesting problem is multi-agent moving-target search (also known as cops and robbers) in which there are several pursuit agents cooperatively trying to capture one or more target agents. This is the main focus of this study. We will assume that the graph is undirected and all agents have perfect information. The emphasis is on adversarial multi-agent moving-target search with a single target. The solutions presented here are easily extendable to multiple targets but, because the performance measures with multiple targets is not as clear-cut as with a single target, we will focus only on the single target case.

This dissertation establishes the thesis that an effective and efficient solution to the multi-agent moving-target search problem can be achieved using a heuristic evaluation function based on *cover sets*. A cover set is the set of locations that the pursuers can "cover" in that the target can not reach that location without being captured. The size of the cover set gives heuristic information about the mobility of the target — the pursuers maximizing the size of the cover set is equivalent to them surrounding the target. A simple algorithm that uses the cover heuristic, CRA, is developed. CRA avoids the exponential growth of the joint action-space when multiple agents are present by maximizing the size of the cover set independently for each pursuer, while introducing coordinated cooperation between pursuers. A target algorithm based on the cover heuristic, TGC, is also developed. This approach improves the state-of-the-art on single- and multi-agent moving-target search for both pursuers and targets.

Also, a non-uniform-speed framework is developed that removes the restrictions on the edge costs and the speeds of the agents. In previous work the edge costs are uniform and the agents traverse one edge per step. We test CRA and TGC on this framework to analyze how pursuers with a speed disadvantage but a numerical advantage fare against a fast target. These studies show that previous approaches to moving-target search are ill-suited for these situations.

Chapter 2 presents theoretical results concerning the capturability of a target in grids. It shows that three pursuers are sometimes necessary for general grid-worlds and that two pursuers are sufficient in an empty grid. These results extend previous results on the number of pursuers necessary and sufficient for capture on graphs. Some results are also presented about the variable-speed framework and the cover heuristic, for instance that the cover heuristic gives optimal strategies for one pursuer chasing one target in graphs where the pursuer has a winning strategy.

An optimal solver is implemented and used as an absolute measure of performance. The optimal strategy can only be calculated for small maps and with a small number of agents due to its time complexity, but the results on these problems can be used to compare the performance of different approaches. CRA is compared to the optimal strategy showing that, in most cases, the success rate of CRA is within a few percentage points of the optimal strategy against an optimal target (for the

maps we use). This is in contrast with previous approaches that fare poorly in this situation. TGC is also compared to the optimal strategy showing that the cover heuristic also gives rise to an almost optimal target in these maps.

Experiments are run on both commercial-game maps and hand-made maps to test the performance of CRA and compare it to previous approaches. It is made evident that CRA does significantly better than all other methods in a wide range of situations. Finally, the run time of the different approaches is compared to analyze practical considerations of CRA.

The thesis is organized as follows: Chapter 1 gives an overview of previous related work, to give insight into the problem and its complexities. Chapter 2 deals with theoretical properties and results to study the limits and properties of the problem. Chapter 3 introduces and explains in detail heuristic methods, in particular the cover heuristic, to show how they can effectively solve the problem. Finally, Chapter 4 gives an empirical evaluation to ascertain the advantages of the methods in a practical context. Parts of this dissertation appear in "A Cover-Based Approach to Multi-Agent Moving Target Pursuit" by Isaza, et al. [19].

Chapter 1

Related Work

There has been relatively little research on multiple-agent moving-target search. In this chapter we explore existing work on single-agent search, single-agent moving-target search and the theoretical results for single- and multiple-agent moving-target search problems. We also review state abstraction methods, as they have been used for single-agent moving-target search and because they will be useful when exploring heuristic methods in Chapter 3. These previous methods serve as a basis for the work presented here and as a comparison point.

1.1 Theoretical properties of pursuit on a graph

Parsons presented the first theoretical results related to the problems of pursuit [29]. He considers the problem of capturing an infinite-speed agent using multiple agents. The solution to this problem requires sweeping the graph so that the target has no possible hiding vertices. Parsons proves several results about the number of pursuers needed for this problem — the search number of the graph — when the graph is a tree. Megiddo, et al., show that determining if the search number of an arbitrary graph is below a given constant is NP-complete [27].

Aigner and Fromme make the first comprehensive study on pursuit, or the game of cops and robbers [2]. They characterize the cop-win graphs — for which the pursuers (cops) have a winning strategy — and the robber-win graphs. The most relevant contributions are the results for more than one pursuer. They prove that, for any value of n, there is a graph that requires at least n pursuers to capture an evading target. They also prove that, for *planar* graphs, 3 cops are always sufficient, that is, they always have a winning strategy. To prove this, they first prove that in planar graphs one pursuer can control a shortest path between two vertices by standing on the path and moving when the target moves so that the target cannot step into that path without being captured. By controlling two separate paths from a vertex v_1 to a vertex v_2 two pursuers can control a region, as long as there is no shorter path outside the region between the vertices. The third pursuer then expands this region by taking control of another path, which frees one of the other pursuers to expand the region again. This process continues until all the vertices in the graph are controlled by the pursuers and

the target is necessarily captured. It is interesting to note that this idea is similar to cover (presented in Chapter 3).

For an example of a planar graph in which three pursuers are necessary and sufficient, consider Figure 1.1. With less than 3 pursuers the target has a winning strategy because there are always 3 edges leading out of any given vertex and all the cycles are of length 5 — the pursuers cannot take short cuts. Three pursuers are sufficient because they can surround the target by standing in each of the neighboring vertices of the target's vertex [2].

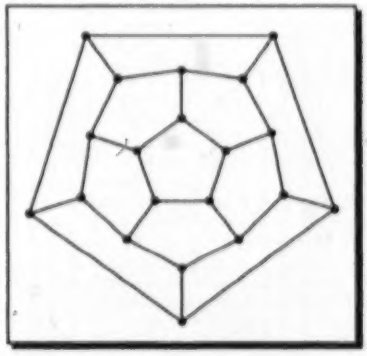


Figure 1.1: A planar graph in which 3 pursuers are necessary to capture a target.

Nisse and Suchan consider the problem of a faster target [28]. They prove that in this situation the number of pursuers necessary in a planar graph becomes unbounded. To prove this, they prove that the strategy that the target should follow consists of keeping an escape margin around so that it can always escape to an adjacent subgrid. They arrive to the result that $\Omega(\sqrt{\log n})$ pursuers are necessary to capture a fast target in an $n \times n$ 4-connected grid.

In arbitrary graphs, the complexity of determining if n pursuers can capture a target is EXPTIME-complete [31]. Goldstein and Reingold prove this result by transforming an alternating Boolean formula (ABF) problem [14]. One pursuer is the prover while the remaining are split into two groups, each representing the variables for each of the two players. Determining if the target can be captured is equivalent to solving the ABF problem, and it can be done in exponential time (Section 2.1), therefore it is EXPTIME-complete. Section 2.1 presents the algorithm to determine if n pursuers can capture a target, and to calculate the optimal strategy.

Bonato, et al. [6], consider the problem of determining the *search-time*, which is the time the pursuers require to capture the target when playing optimally (against an optimal target). The search-time does not depend on the stating locations — it is an invariant of the graph. Bonato, et al., prove that deciding if n pursuers can capture a target within a given time limit is NP-complete. This result demonstrates the fact that determining the number of pursuers that should be used in a particular problem is not trivial.

1.2 Single-Agent Search

Single-agent search is an important problem in AI and it is a good starting point when analyzing moving-target search. Several approaches to moving-target search problems are based on single-agent search techniques. For this reason, existing solutions to the single-agent search problem are introduced before moving on to moving-target search.

The single-agent search problem takes place on a weighted graph $G = (V, E)$. The set of neighbors of a vertex $v \in V$ is denoted $N(v) \subseteq V$. The cost of going from vertex v to vertex $v' \in N(v)$ is $c(v, v') \in \mathbb{R}^+$. An agent (the problem solver) occupies vertex $v_a \in V$, and the goal is vertex $v_g \in V$. The problem consists of finding a shortest (minimum-cost) path from v_a to v_g.

The most widely-known algorithm for solving this problem is A* [16], which is optimal and complete but too slow and memory-consuming in many situations. The A* algorithm does a best-first search by prioritizing node expansion using a heuristic evaluation function $f : V \to \mathbb{R}^+$. The evaluation function for a vertex $v \in V$ is defined as $f(v) = g(v) + h(v)$ where $g(v)$ is the cost of the shortest path from the start vertex (v_a) to v, and $h(v)$ is an estimate of the cost of the shortest path from v to v_g. If h is admissible (i.e., it never overestimates the true shortest-path cost), then A* is guaranteed to return a shortest path between v_a and v_g.

There are many variations of A* best suited for specific problems. Some of them are Iterative-Deepening A* (IDA*) which reduces the space complexity [24], non-optimal real-time variants like Real-Time A* (RTA*) and Learning Real-Time A* (LRTA*) which can interleave planning and execution [25], non-optimal variants like Partial-Refinement A* (PRA*) which compromise optimality for solution time [33], and many more. The single-agent search problem has been studied extensively.

1.3 Moving-Target Search

The term *moving-target search* was coined by Ishida and Korf, who developed an algorithm with the same name, MTS [22], which was the first attempt at solving this problem. It is, basically, an extension of the single-agent search algorithm Learning Real-Time A* (LRTA*) [25], which, as its name implies, is a learning and real-time extension of A* [16].

The idea behind LRTA* is to interleave planning and execution so that the search algorithm takes a constant time per move. To achieve this, in each step, LRTA* chooses the successor with the smallest f value instead of doing a full A* search to the goal. It also keeps a table with learned heuristic values. Each time a heuristic depression is found — a vertex that has a lower heuristic value than every neighbor — LRTA* updates the heuristic values to "fill in" the depression. The pseudo-code for this algorithm is shown in Figure 1.2. It takes the current vertex, v, and returns the neighbor, $v' \in N(v)$, with the smallest f value.

While LRTA* learns the heuristic values for all states given a fixed goal, MTS learns heuristic

LRTA*$(v) : v' \in N(v)$
1 **for each** $v' \in N(v)$ **do**
2 $\quad f(v') \leftarrow c(v, v') + h(v')$
3 **end**
4 $h(v) \leftarrow \min_{v'} f(v')$
5 **return** $\arg\min_{v'} f(v')$

Figure 1.2: LRTA* algorithm

values for every pair of states. Also, MTS learns when the target moves, not only when the pursuer moves. To do this, MTS keeps a table $h(v, w)$ of heuristic values between vertices v and w. Figure 1.3 shows the details of the algorithm.

MTS$(v, w) : v' \in N(v)$
1 $h(v, w) \leftarrow \max\{h(v, w), \min_{v'}[h(v', w) + c(v, v')]\}$
2 **return** $\arg\min_{v'} h(v', w)$

MTS-TargetMove(v, w, w')
1 $h(v, w) \leftarrow \max\{h(v, w), h(v, w') - c(w, w')\}$

Figure 1.3: MTS algorithm

Because MTS learns the heuristic values for the current position of the target, every time the target moves it must learn the heuristic values from scratch (for the new target's position). This is the most serious drawback of this algorithm. The second version of MTS (MTS2) tries to mitigate this problem using "commitment" and "deliberation" [23]. The idea of commitment is to aim for a goal vertex while in a heuristic depression, ignoring the target's moves. The degree of commitment, $C \in \mathbb{N}$, specifies for how long to fix the goal after the border of the depression is reached. By doing this, the heuristic values in a depression can be filled-in only once even when the target moves. Once outside the depression the normal operation resumes. Deliberation deals with the problem of having to fill in a depression by visiting every state in the depression. With deliberation, when the pursuer finds itself inside a depression, instead of choosing an action and updating a value, it performs an offline search — without executing any moves — until the depression border is reached, it then performs the update, and starts following the path to exit the depression. The degree of deliberation, $D \in \mathbb{N}$, specifies for how long to do the offline search after finding an edge of the depression.

Figure 1.4 shows the MTS2 algorithm. **MTS2**(v, z, C, D) is called when the pursuer is to move from vertex $v \in V$ and the current goal vertex is $z \in V$. C and D are the degree of commitment and the degree of deliberation, respectively. The current commitment $c \leq C$ keeps track of how committed the agent is to the current goal vertex; it starts at 0. **MTS2-TargetMove**(v, w, w') is called when the target moves from $w \in V$ to $w' \in N(w)$. Here $f(v', w)$ is defined as the heuristic value of v' plus the cost of going from v to v': $f(v', w) = h(v', w) + c(v, v')$. This differs from the original version, in which $h(v', w)$ is used alone to test for depression, but using $f(v', w)$ allows the algorithm to handle non-uniform costs. When the costs are uniform ($c(v, v') = 1, \forall v' \in N(v)$) the

MTS2$(v, z, C, D) : v' \in N(v)$

```
1  if there is an offline path then                          check for previous offline path
2      v' ← pop the first vertex on the path p
3      return v'
4  end if
5  f_min ← min_{v'} f(v', z)
6  if h(v, z) ≥ f_min then c ← max{c − 1, 0}                 decrease commitment
7  if h(v, z) < f_min then c ← C                             commit to the current goal
8  if h(v, z) < f_min ∧ D ≠ 0 then
9      p ← OfflineSearch(v, z, D)                            do offline search
10     v' ← pop the first vertex on the path p
11     return v'
12 end if
13 if h(v, z) ≥ f_min ∨ D = 0 then
14     h(v, z) ← max {h(v, z), f_min}                        update heuristic estimate
15     return arg min_{v'} f(v', z)
16 end if
```

MTS2-TargetMove(v, w, w')

```
1  if c = 0 ∨ v = w then                                     when there is no commitment
2      h(v, w) ← max {h(v, w), h(v, w') − c(w, w')}           update heuristic estimate
3      z ← w'                                                update the goal vertex
4  end if
```

OfflineSearch(v, w, D) : path

```
1  q ← {[v, 0, h(v, w)]}                                     priority queue
2  s ← ∅                                                     closed set
3  f_max ← 0                                                 maximum f value, the border
4  while q is not empty and d ≠ 0 do
5      [u, g, h] ← pop element with smallest g + h from q
6      s ← s ∪ {u}                                           add to the closed set
7      if u = w then                                         target found
8          return path leading to w
9      end if
10     f_min ← ∞
11     for every neighbor n ∈ N(u) not in s do               expand neighbors
12         q ← q ∪ {[n, g + c(u, n), h(n, w)]}
13         if g + c(u, n) + h(n, w) < f_min then
14             f_min ← (g + c(u, n)) + h(n, w)               keep track of the smallest f
15         end if
16     end for
17     if g + h ≥ f_min then                                 outside depression
18         if g + h > f_max then                             border found
19             f_max ← g + h
20             g_b ← g                                       keep the g value of the border node
21             z ← u                                         this is the new goal vertex
22             for x ∈ s do h(x, w) ← h end for              update heuristic estimates
23         end if
24         if border found and g − g_b > D then              maximum depth reached
25             return path leading to z
26         end if
27     end if
28 end while
29 return error, the target is not reachable
```

Figure 1.4: The second version of the MTS algorithm

algorithm reduces to the original version [23, 21]. In the original version "the unit cost assumption is adopted to simplify the definition of speed of the problem solver and the target" [21, Section 3.2]. Later we will consider non-uniform speeds so it is necessary to generalize the algorithm.

MTS2 differs from the initial version of MTS: when it is the pursuer's turn to move MTS2 checks if the agent is inside a heuristic depression and if so commits to the current target's vertex (Line 7). Once outside a depression the commitment decreases gradually (Line 6). Then either an offline search or a real-time search are performed (Lines 8 to 16). When it is a target's move we perform the update as before only if the pursuer is not committed or the target has been reached.

The offline search consists of a best-first search starting from the pursuer's vertex, v, until reaching the border of the depression or the target, w. If the border is reached before finding the target then the search continues for D steps, to ensure that the depression is actually overcome (Lines 24 to 26). Each time a border with a larger f value is found, the heuristic estimates of the vertices in the closed list are updated (Line 22).

MTS2 has considerable advantages over MTS. While MTS always converges to the correct heuristic values given enough time, MTS2 manages to catch the target faster with a good choice of D and C and it converges faster as well [23]. Nonetheless, it still presents problems: the parameters D and C have to be tuned to specific targets and, most importantly, it fails to minimize the target's mobility. Finally, it does not coordinate multiple pursuers.

1.4 State-space abstraction

State-space abstraction is a technique used to speed up search [9, 10, 17, 33]. It has been used extensively in several domains, from reinforcement learning [5] to stochastic shortest path problems [20]. For single-agent search it can be used either by searching in a smaller abstract space [10, 33], or by using the abstraction as a heuristic estimate in the non-abstract space [18]. For our purposes, the relevant approach is searching in an abstract space, or *abstract graph*. The basic idea is to generate an abstract graph from the original, *ground-level*, graph. The abstract graph is smaller but keeps the overall structure of the ground-level graph. Search is then performed in this abstract graph.

A common abstraction algorithm is clique abstraction [9, 33]. Unlike other approaches that rely on domain knowledge to build the abstraction [7], clique abstraction relies on the local structure of the graph to automatically build abstractions. It clusters *cliques* into abstract states so that all the ground states are one action away from every other ground state in the clique. If there are orphaned states clique abstraction merges them into an adjacent abstract state. This way the abstract states consist of localized groups of ground states.

Once the abstract graph is generated, any search algorithm can be used on it to generate an *abstract path*. After an abstract path is found it is *refined* to generate a ground-level path. The refinement process consists of building a *corridor* with all the ground states that abstract to any of the abstract states that lie on the abstract path. Then a search algorithm is run inside this corridor to

generate the ground-level path.

An advantage of this method is that the abstract path can be refined by parts as needed. When a ground level action is needed the path is refined from the current abstract state to the next so that only a few ground actions are generated. This partial refinement process distributes the computation time throughout the execution so that the first-move time is reduced. It also saves time when the complete path is not needed, for example when the target is moving.

A viable approach to the moving-target search problem is to follow the shortest path from the current vertex to the target's vertex. The problem with this approach is that it is expensive to calculate a path on every move. Abstraction can be used to speed-up the path finding to obtain an approach that is similar to MTS but does not learn, one such path-finding algorithm is Partial Refinement A* (PRA*) [33]. PRA* uses the abstraction and refinement techniques just described to generate partial paths in a short time, we are interested in PRA*(1) which refines only one abstract action at a time. It can be run on every move without much loss because an abstract path is generated and only one abstract action is refined. PRA* automatically chooses the level ℓ_m that is half-way between the ground level and the level at which the source and destination vertices lie on the same abstract state to execute an A* search, with $\ell_m \leq \frac{\log_k |V|}{2}$, where k is the average size of the cliques. The maximum size of the abstract graph at level ℓ_m is $\sqrt{|V|}$ and the complexity of refining the path is the number of ground states corresponding to the origin and destination abstract states ($2k$) times the number of levels for which this is done (ℓ_m): $O(2k\ell_m) = O(k \log_k |V|)$. Therefore, the complexity per move of PRA* is the complexity of finding a path in the abstrat graph ($O(\sqrt{|V|} \log \sqrt{|V|})$) plus the complexity of refining one abstract action ($O(k \log_k |V|)$), assuming the branching factor is constant.

Another approach is to use MTS with abstraction. Bulitko and Sturtevant use this idea in Partial Refinement MTS (PR MTS) [8], where they run MTS in an abstract space and then refine the abstract action. They report improvements in convergence time over the non-abstracted MTS. Nonetheless, these algorithms still suffer from the problems present on MTS: they do not minimize the target's mobility and they do not coordinate multiple pursuers.

1.5 Other approaches

The amount of literature on practical approaches to moving-target search problems is small. The most widely known approach is MTS and the related algorithms that aim for the current location of the target and treat the problem as a single-agent search problem. This approach has already been discussed and its drawbacks outlined. Another approach is the Trailblazer algorithm which is more a trailing algorithm than a pursuit algorithm [11]; while it deals with coordination it does not try to reduce the target's mobility or in any way surround it.

The one algorithm that deals with pursuer coordination and with partially observable environments is Multiple Agents Moving Target Search (MAMTS) [13]. The MAMTS algorithm keeps a

belief set for every pursuer so that when the target is not visible they can coordinate the pursuit by choosing different regions to search for the target. Once the target is spotted they fall back to the "shortest path to target" strategy, which has drawbacks that have been discussed already.

In summary, all existing approaches try to capture the target by following a short route to its current position. This approach is effective against targets that are not aware of the pursuer or that have simple strategies but it is ineffective against more involved targets, as will be shown in Chapter 4. An effective pursuit algorithm needs to reduce the target's mobility and, if multiple pursuers are available, surround the target. A good pursuer will *outsmart* its target rather than outrunning it, as is often seen in nature.

Chapter 2

Theoretical Properties

Research on moving-target search (also known as pursuit games or cops and robber) has focused on uniform-cost graphs with uniform-speed agents (i.e., traveling one edge per move). The uniform-speed problem is considered first. In this context we present some interesting results for grids. Uniform speed moving-target search is an interesting problem but in real-world scenarios agents have different speeds and the terrain is not uniform. We therefore define and analyze the more realistic non-uniform-speed problem as well.

2.1 Uniform speed

For now assume that all agents have the same speed, that is, they traverse one edge per time step. The game takes place on an undirected graph $G = (V, E)$. There are $n + m$ agents: n pursuers $(a_1, \ldots, a_n)$ and m targets $(a_{n+1}, \ldots, a_{n+m})$. Each agent a_i, for $1 \leq i \leq n + m$, has an associated vertex at time step t: $v(a_i, t)$, starting at $v(a_i, 0)$.

The game proceeds as follows. First the targets move, each choosing a vertex $u \in N[v(a_i, t)]$, $v(a_i, t+1) = u$, then the pursuers move. The process repeats, removing targets as they are captured, until there are no targets left. A target a_t is captured at step t_c by a pursuer a_p if $v(a_t, t_c) = v(a_p, t_c)$ or $v(a_t, t_c) = v(a_p, t_c - 1)$.

The problem of determining if n pursuers can capture a target can be solved using a backtracking algorithm as shown in Figure 2.1. The algorithm starts from the capture joint-states and goes back to determine the capturability, and the time-to-capture, of all possible joint-states. To do this, it starts assigning a value of 0 to all the joint-states in which the target is captured and pushing them into a priority queue (Lines 4 to 6). After popping a joint-state from the queue, each of its predecessors is added to the queue after its time-to-capture ($h_{\{p,t\}}(s)$) is calculated (Lines 15 to 26). The process repeats until the queue is empty. There are $|V|^{n+m}$ joint states and there are at most b^{n+m} joint predecessors if b is the number of moves available for each agent. Calculating the targets' value takes $O(b^m)$ so the algorithm takes $O(|V|^{n+m} b^{n+2m})$ operations in the worst case. Given that each joint state has to be visited at least once, the time complexity is $\Omega(|V|^{n+m})$.

```
OptimalStrategy()
 1  for all joint-states, s do
 2      h_p(s) ← ∞
 3      h_t(s) ← ∞
 4      if the target is captured in s then
 5          push [s, t = target-to-move, h = 0] into priority queue q
 6      end if
 7  end for
 8  while q is not empty do
 9      (s, t, h) ← pop the element with smallest h from q
10      if t = target-to-move then
11          h_t(s) ← h
12      else
13          h_p(s) ← h
14      end if
15      for every successor s' of s do
16          if t = target-to-move then
17              if h_p(s') = ∞ then
18                  push (s', pursuer-to-move, h + 1) into q
19              end if
20          else
21              if h_t(s') = ∞ then
22                  h' ← TargetsValue(s')
23                  push (s', target-to-move, h') into q
24              end if
25          end if
26      end for
27  end while

TargetsValue(s) : value
 1  h ← 0
 2  for every successor s' of s do
 3      if h_p(s') = ∞ then
 4          return ∞
 5      end if
 6      if h_p(s') > h then
 7          h ← h_p(s')
 8      end if
 9  end for
10  return h
```

Figure 2.1: The algorithm to calculate the optimal strategies and to determine the capturability of the target for all starting configurations. Complexity: $O(|V|^{n+m}b^{n+2m})$ where b is the branching factor.

This algorithm is similar to Dijkstra's algorithm, the only difference is the fact that it has to switch between calculating a value when the pursuers are to move and calculating it when the target is to move. While in the first case it is straightforward (line 18), for the second (line 22) it is necessary to generate all successors for the target: if there is one destination state, s, where $h_t(s) = \infty$, then the value cannot be determined (yet). Otherwise, the value is the largest of the successor values (see **TargetsValue**(s) in Figure 2.1).

After the backtracking algorithm terminated, joint-states for which $h_t(s) = \infty$ are the joint-states in which the target cannot be captured. If there are no such joint-states then the target can always be captured and we know that n pursuers are sufficient. Furthermore, we know the strategy: pursuers greedily choose the successor with the smallest $h_t(s)$ and targets choose the successor with the smallest $h_p(s)$.

There are situations where we know, using theoretical results, how many pursuers are necessary or sufficient without running the algorithm. For instance, in planar graphs at most three pursuers are needed.

Theorem 2.1.1 ([2, Theorem 6]). *Three pursuers are always sufficient to capture a single target in any planar graph.*

See Section 1.1 for an outline of this proof.

Theorem 2.1.1 is relevant for many applications, including video games, in which planar graphs are used extensively. There are further results on different types of graphs, which may be of use for some particular applications [2, 3, 4, 30]. For now we concentrate on planar graphs. An interesting type of planar graph is the 4-connected grid, which is popular because of its simplicity. Being a planar graph, from the previous result we know that three pursuers are sufficient, but another matter is if they are necessary. We will first consider an empty rectangular grid.

Theorem 2.1.2. *In an empty rectangular grid two pursuers are necessary and sufficient to capture a single target [15].*

Proof. The grid is of size $C \times R$, where C is the number of columns and R is the number of rows. There is one target, T, and two pursuers, P and Q. Suppose that at time t the target is at column $col(T, t)$ and row $row(T, t)$, similarly with the pursuers. In at most $\max\{C, R\}$ steps the pursuers move so that $col(P, t_0) = col(T, t_0)$ and $row(Q, t_0) = row(T, t_0)$, with $t_0 \leq \max\{C, R\}$. From then on, P stays in the same column as T and Q stays in the same row as T. That is, at time t:

1. If the target stands still both pursuers move closer while $col(P, t+1) = col(T, t+1)$ and $row(Q, t+1) = row(T, t+1)$. The distance decreases for both pursuers.

2. If the target changes row then Q changes row as well so that $row(Q, t+1) = row(T, t+1)$, and P changes row so that either it is closer to the target or the target is closer to the edge, while $col(P, t+1) = col(T, t) = col(T, t+1)$.

3. If the target changes column then P changes column as well so that $col(Q, t+1) = col(T, t+1)$, and Q changes column so that it is closer to the target or the target is closer to the edge, while $row(P, t+1) = row(T, t) = row(T, t+1)$.

By following this strategy the pursuers will eventually corner the target and capture it, showing that two pursuers are sufficient. It is easy to see that one pursuer is not sufficient by noting that in rectangular grids all vertices have at least two neighbors (assuming $C > 1$ and $R > 1$) and there are no cycles of lenght less than 4, which proves that two pursuers are necessary. □

This proof may suggest that two pursuers are sufficient for any type of grid-world but this is not the case. The following theorem establishes that three pursuers may still be needed in arbitrary grid-worlds.

Theorem 2.1.3. *Two pursuers are not always sufficient in an arbitrary grid-world.*

Proof. This will be proved by constructing a grid-world in which, for some initial configuration, two pursuers cannot capture a target. The graph in Figure 1.1 may be the simplest planar graph in which two pursuers are not sufficient [2]. That graph can be transformed into a grid-world as in Figure 2.2. Each of the 20 vertices of the graph in Figure 1.1 corresponds to a numbered node in the grid-world in Figure 2.2. Each numbered node has three "adjacent" numbered nodes, always 24 vertices away with no bifurcations in between. Assume that we place each of the agents in a different numbered node. The target can move to any of the three "adjacent" nodes in the time it takes the pursuers to reach the node in which the target was standing. Because there are always three "adjacent" nodes and there are no cycles of length less than 5×24, the target can always place itself in a node without meeting a pursuer so that the situation remains the same.[1] □

The **OptimalStrategy** algorithm was executed on this map (Figure 2.2), producing the expected result. But notice that there are some positions in which the target cannot escape with only two agents, like when the target is in the middle of a corridor surrounded by the pursuers. More specifically we have the following result:

Proposition 2.1.1. *The target can escape in the grid-world in Figure 2.2 if and only if it can get to a node n before any pursuer can reach n.*

Proof. The **OptimalStrategy** algorithm was run on the map and the proposition was verified for all joint states. □

These results are useful for determining how many pursuers to use in particular situations. They are also a good starting point when determining which strategy to use: if there are enough pursuers in a given graph then they can use a conservative strategy that minimizes the worst-case capture

[1] For the more general proof that, for any n, there is a graph in which n pursuers are necessary, see Theorem 3 of Aigner and Fromme, 1984 [2].

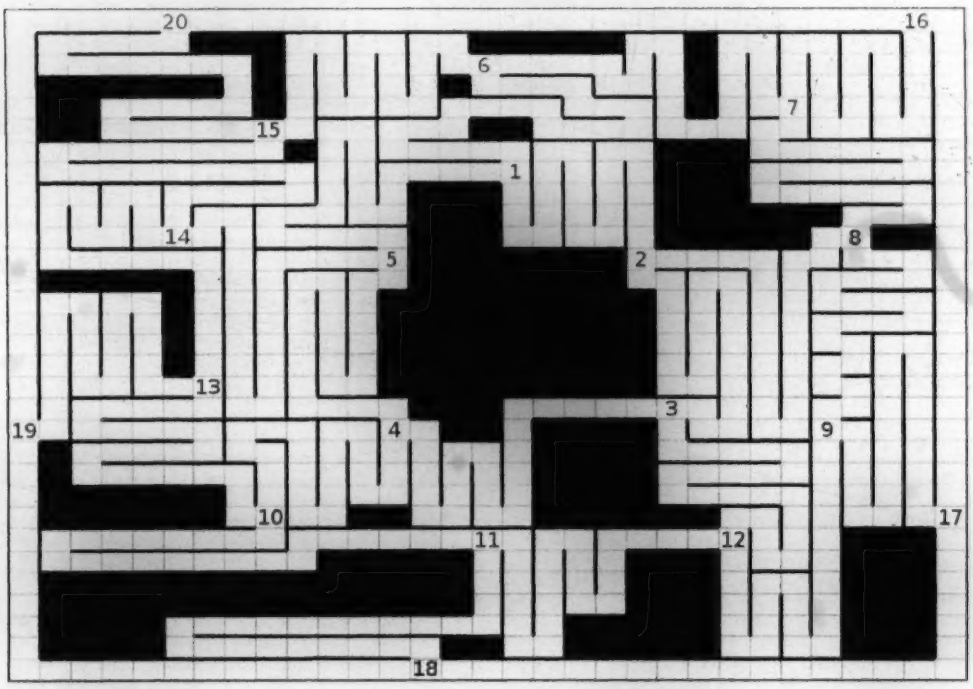


Figure 2.2: A grid-world in which three pursuers are necessary.

time. If, on the other hand, we know beforehand that there is no winning strategy for the pursuers they might as well hope that the target will make mistakes and use a risky strategy that just rushes towards the target.

2.2 Non-uniform speed

Uniform-speed problems, while theoretically interesting, are not realistic. Here a model is introduced for the moving-target search game with non-uniform-speed agents on non-uniform graphs, which is a generalization of the simpler model presented in the previous section.

The game takes place in an undirected weighted graph $G = (V, E)$ with edge costs $c : V \times V \to \mathbb{R}^+$, where $c(u, v)$ is the minimum cost of going from vertex u to vertex v. There are $n + m$ agents: n pursuers $(a_1, \ldots, a_n)$ and m targets $(a_{n+1}, \ldots, a_{n+m})$. Each agent a_i, for $1 \leq i \leq n + m$, has speed $s_i > 0$, which remains constant. Each agent has a number of properties that change each time they execute an action: two vertices u_i and v_i, and a time t_i, to mean that the agent reaches vertex v_i at time t_i, after leaving vertex u_i. Initially, agents are at the starting vertices $u_i = v_i = v_{i_0}$ at time $t_i = 0$.

The game proceeds as follows. Agents with the lowest value of t_i are chosen to move next, ties are broken by sorting the agents so that agent a_i goes before agent a_j when $i < j$, and targets always go before pursuers. Each agent a_i either (1) chooses a neighbor w of v_i, increments its time by the

time cost, $t_i \leftarrow t_i + \frac{c(v_i, w)}{s_i}$, and sets $u_i \leftarrow v_i$ and $v_i \leftarrow w$, or (2) chooses to stand still, sets its time t_i to the minimum time of the agents of the opposing side or increments it by δ, a parameter, and sets $u_i \leftarrow v_i$. The process repeats, removing targets as they are captured, until there are no targets left or there is a timeout.

A target a_t is captured by a pursuer a_p when:

1. The target traverses an edge (u_t, v_t) while the pursuer traverses the same edge in the opposite direction: $v_t = u_p$, $u_t = v_p$, $t_t \geq t_p - c(u_p, v_p)/s_p$ and $t_p \geq t_t - c(u_t, v_t)/s_t$
2. The target traverses an edge while the pursuer overtakes it: $u_t = u_p$, $v_t = v_p$, $t_t - c(u_t, v_t)/s_t \leq t_p - c(u_p, v_p)/s_p$ and $t_t \geq t_p$
3. The pursuer traverses an edge while the target overtakes it: $u_t = u_p$, $v_t = v_p$, $t_p - c(u_p, v_p)/s_p \leq t_t - c(u_t, v_t)/s_t$ and $t_p \geq t_t$
4. The target reaches a vertex in which the pursuer is standing: $v_t = u_p = v_p$ and $t_t \leq t_p$
5. The pursuer reaches the vertex in which the target is standing: $u_t = v_t = v_p$ and $t_t \geq t_p$
6. The target reaches a vertex at the same time as the pursuer: $v_t = v_p$ and $t_t = t_p$

Notice that agents will be unable to change their mind after executing an action before reaching the destination vertex. If the action takes long to execute, either because the cost of the action is high or because the speed of the agent is low, the agent will be unable to take action for a long time. In other words the agent commits to the action for as long as it takes. This may cause problems when there are other agents that can quickly go between vertices because the slow agent will have less chances of moving and has to carefully decide before committing. One way of avoiding this is by having similar costs in all actions and having similar speeds for all agents.

A point that also needs some attention is deciding who moves first. Because of the asymmetry of pursuit games, the target has to move first. To see why this is the case suppose that a pursuer and the target start in adjacent vertices. If the target were to move second then the pursuer would move to capture while the target would not get a chance to move, which is unfair. If, on the other hand, the target moves first then it always gets a chance to move on the round it is captured. In other words, having the target move first ensures that it always gets the chance to move before the capture condition is evaluated.

That is the turn-based version. In the simultaneous move case there is no need to choose a player to move first. Simultaneous games are defined as games in which both players have to move without knowledge of the opponent's move. In most games this is achieved by making the players choose the moves simultaneously, hence the name. In the game just described the moves can not be simultaneous because the game is asynchronous — the move time depends on how long each agent takes to go from u_i to v_i. Therefore, to make the game "simultaneous", we have to hide information

about the last move. To do this we only reveal an agent's position when it reaches a vertex. In other words, the agent is hidden while traversing edges.

We can take an intermediate stance between turn-based and simultaneous games in which information is not given immediately but is given earlier than in the simultaneous case just described. For example we can define a fixed time interval in which to divulge information. In this case agents could *pay* for information by standing still for the fixed interval and getting the information about opponents' moves before deciding on a move themselves. The study of this intermediate kind of games if left as a subject for future research.

For now lets consider the turn-based version. It would be desirable to extend the uniform-speed results to this domain. The following theorems show that extending the results is not straightforward.

Theorem 2.2.1. *The fact that n pursuers of speed s can capture a target of the same speed in a graph G does not imply that the same n pursuers in the same graph but with speed $s - \epsilon$ can capture the target, for an arbitrarily small value of ϵ.*

Proof. Consider the map in Figure 2.3, with a target in the lower left corner and a pursuer in the lower right corner. If the pursuer and target have the same speed then the target will not be able to get past the middle cell (capture condition 6). If, on the other hand, the speed of the target is slightly larger it will get into the corridor leading to the cycle and thus will escape capture. □

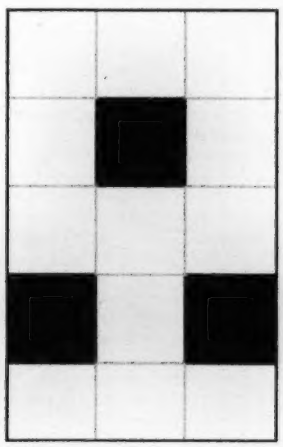


Figure 2.3: A grid map in which a target can escape from a pursuer.

Theorem 2.2.2 ([28, Theorem 2])**.** *The number of pursuers necessary to capture a faster target in a planar graph is unbounded.*

Theorems 2.2.1 and 2.2.2 are show-stoppers. The only bound available is the one by Nisse and Suchan which says that at least $\sqrt{\log n}$ pursuers are necessary in an empty $n \times n$ grid [28]. In the

extreme case where the target has infinite speed, graph-sweeping methods must be used [29, 27]. To sweep the graph, the pursuers move through the graph blocking out regions so that the target can not get to those regions without meeting a pursuer.

The number of pursuers necessary in these cases will not be considered further. Instead, the variable speed framework will be used to test the performance of the algorithms and experimental results will be given. These results support the fact that a small number of pursuers will not be able to capture the target in every situation. As the speed of the pursuers decreases, the number of pursuers necessary increases super-linearly. The number of extra pursuers necessary depends on the graph, this will be further explored in Section 4.2.

Chapter 3

Heuristic Methods

Heuristic methods are used when a complete solution to a problem is not feasible; they give guidance to the problem solver so that it follows a path that is likely to lead toward the solution. In search problems, a *heuristic function* can be used to find near-optimal solutions by doing a best-first search or to speed up the search time when finding an optimal solution with, for example, A* [16]. A heuristic function gives an estimate of the optimal solution cost.

In one extreme, a perfect heuristic function gives the exact solution cost for every state. In the other extreme, the totally uninformed heuristic gives no guidance. There is normally a trade-off between the time required to calculate a heuristic and its quality; heuristic functions normally simplify the problem so that they are fast to calculate, at the cost of quality. It would be desirable to have a heuristic function that gives almost optimal estimates with as little computation as possible.

One of the most common heuristic functions used in single-agent search is geometric distance (e.g., Manhattan or Euclidean distance). A first approach to the moving-agent problems could consist of using geometric distance as the heuristic estimate. It may be surprising that doing this can lead to poor solution quality, which is not expected given the success of the geometric distance heuristic in common single-agent search problems. This is the major drawback of MTS and related algorithms.

In this chapter we explore this counter-intuitive behavior and present a more appropriate solution: cover. Cover performs exceptionally well in a wide variety of situations, although being considerably more expensive to calculate than geometric distance. An algorithm that builds on the strengths of the cover heuristic is presented, after the concept of cover has been introduced.

3.1 The geometric distance heuristic

Geometric distance is a good way to approximate the solution to some single-agent search problems, especially those on planar graphs. It has been used extensively in various domains with good results [16, 26]. The geometric distance heuristic has the advantage of being fast to calculate and being *admissible*. Admissible heuristic functions never overestimate the solution cost, which is required

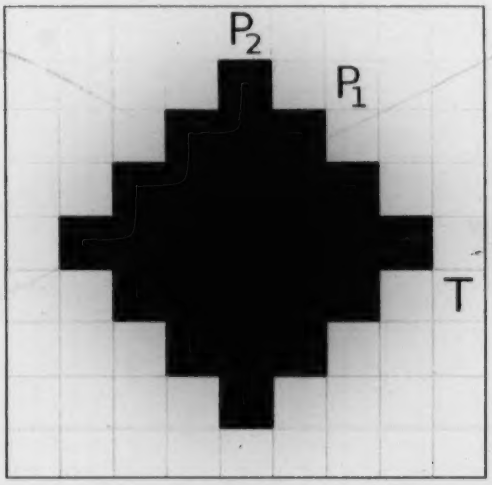


Figure 3.1: A map in which a geometric distance heuristic does a bad job of capturing a target.

when using A* to get an optimal solution the first time a goal state is expanded.

In four-connected grids Euclidean distance may not be as good as Manhattan distance, which gives the actual solution costs when there are no obstacles. Manhattan distance is defined as

$$d_4\left(\langle x_1, y_1\rangle, \langle x_2, y_2\rangle\right) = |x_2 - x_1| + |y_2 - y_1|, \tag{3.1}$$

it is the distance it would take you to go from one point to another inside a city in which the streets form a grid (therefore the name). This heuristic gives an exact solution cost in empty grids, and a reasonable solution cost for grid-worlds in which the obstacle density is low. Similarly, in eight-connected grids the octile distance,

$$d_8\left(\langle x_1, y_1\rangle, \langle x_2, y_2\rangle\right) = \left(\sqrt{2} - 1\right) \min\{|x_2 - x_1|, |y_2 - y_1|\} + \max\{|x_2 - x_1|, |y_2 - y_1|\}, \tag{3.2}$$

gives actual costs when there are no obstacles.

It is tempting to use a geometric distance heuristic for the moving-agent search problem. After all, the pursuers want to get as close as possible to the target, that is, to minimize the distance. The problem with this approach is that the mobility of the target is not minimized; the pursuer will, most likely, simply chase the target without being able to corner it. This problem is most obvious when there are multiple pursuers as they will cluster and follow the target as one, instead of using their advantage by surrounding the target. Moreover, this is assuming that the geometric distance heuristic is approximately accurate; if there are many obstacles the behavior only gets worse.

Consider, for example, the round-table map in Figure 3.1 with two pursuers and one target. If the pursuers start close together and use a geometric distance heuristic they will run around taking the shortest route, which will likely be the same for both. This is clearly a bad strategy — a better strategy is to split up and go in opposite directions, even if this means that one pursuer has to take the long route.

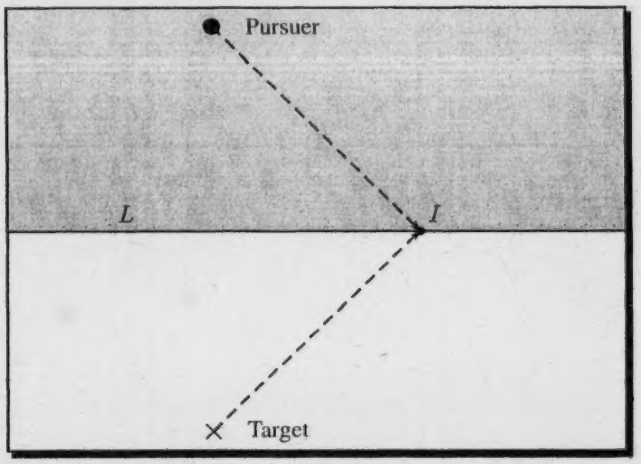


Figure 3.2: The set of points to which both agents can get to at the same time defines a line that separates the pursuer's territory (gray) from the target's territory (white).

3.2 The cover heuristic

To motivate the cover heuristic, consider one target and one pursuer in an empty Euclidean plane. The agents have the same speed, s, the pursuer is at position (x_p, y_p) and the target is at position (x_t, y_t) so that the Euclidean distance between them is

$$d = \sqrt{(x_t - x_p)^2 + (y_t - y_p)^2}. \tag{3.3}$$

In an attempt to find a strategy for the pursuers we consider all the points, L, which the pursuer and the target reach at the same time, where time is the distance divided by the speed:

$$\begin{aligned} L &= \left\{ (x,y) \mid \frac{\sqrt{(x-x_p)^2 + (y-y_p)^2}}{s} = \frac{\sqrt{(x-x_t)^2 + (y-y_t)^2}}{s} \right\} \\ &= \left\{ (x,y) \mid (x-x_p)^2 + (y-y_p)^2 = (x-x_t)^2 + (y-y_t)^2 \right\} \\ &= \left\{ (x,y) \mid (x_p^2 - x_t^2) + (y_p^2 - y_t^2) - 2x(x_p - x_t) - 2y(y_p - y_t) = 0 \right\} \end{aligned}$$

Because the speeds are the same, the set is the perpendicular bisector of the line segment joining the agents, as shown in Figure 3.2. The strategy is straightforward: if the target's trajectory is going to intersect L the pursuer has to aim for the intersection point, I. By definition of L, the pursuer will catch the target at exactly I (unless the target steers away). If the target's trajectory does not intersect L in the future then the pursuer has to go in the same direction as the target so that the distance between them stays the same. The moment the target changes direction the pursuer updates its direction too to follow the above strategy. From this, the target's strategy is also clear: keep the distance between them constant by running away.

If we add another pursuer then there are two lines, L_1 and L_2, one for each pursuer. The strategy for the target is to choose a direction so that its trajectory does not intersect either L_1 or L_2. With

only two lines this is still possible, so the target can still escape (see Theorem 3.2.1). If there are three pursuers then the pursuers can effectively surround the target so that, no matter what direction the target chooses, its trajectory will intersect a line.

Theorem 3.2.1. *On the entire Euclidean plane where the agents can move in any direction, there is no starting configuration (excluding starting configurations in which the target is captured) in which less than three pursuers can capture a target of the same speed.*

Proof. Each pursuer contributes a line to enclose the target and there is no closed polygon with less than three sides. Therefore, three pursuers are needed to surround the target. □

Theorem 3.2.2. *On the entire Euclidean plane, for any number of pursuers, n, there is always a starting configuration in which the target can escape.*

Proof. If the n pursuers are positioned so that their lines do not surround the target, the target can choose a trajectory in which the pursuers will not be able to capture the target. □

This gives an insight into the capture condition and the strategy for both target and pursuers. The lines separate the territory "controlled" by the pursuers from the territory of the target; see Figure 3.2. We will call the pursuer's territory the *cover set*, C, which is the set of all states that the pursuers can reach earlier than or at the same time as the target. The strategies are simple: the pursuers select the joint actions leading to the states that maximize $|\mathcal{C}|$ while the target tries to minimize $|\mathcal{C}|$.

One last theorem deals with a finite Euclidean plane.

Theorem 3.2.3. *On a finite convex subset of the Euclidean plane one pursuer can always capture one target of the same speed.*

Proof. By choosing the direction that minimizes the distance to the target, the pursuer can reduce the distance to the target when the target gets to the boundary of the region. Eventually the distance will become zero and the target will be captured. □

3.2.1 Non-uniform speed

The idea can be extended to the case where the agents have non-uniform speeds. We need to find the set of points that the pursuers can reach earlier than the targets. For each pair of pursuer p and target t, we have

$$\mathcal{C}_{pt} = \left\{ (x,y) \mid \frac{\sqrt{(x_p - x)^2 + (y_p - y)^2}}{s_p} \leq \frac{\sqrt{(x_t - x)^2 + (y_t - y)^2}}{s_t} \right\} \tag{3.4}$$

where the position of the pursuer (resp. target) is (x_p, y_p) (resp. (x_t, y_t)) and its speed is s_p (resp. s_t). The boundary of $\mathcal{C}_{pt}$ is defined by an Apolloninan circle, but we will develop the details anyway. To find the boundary of $\mathcal{C}_{pt}$ we substitute the inequality for an equality and simplify the expression:

$$s_t^2[(x_p - x)^2 + (y_p - y)^2] = s_p^2[(x_t - x)^2 + (y_t - y)^2]$$

Expanding the squares and rearranging we get

$$s_t^2(x_p^2 - 2x_p x + x^2 + y_p^2 - 2y_p y + y^2) - s_p^2(x_t^2 - 2x_t x + x^2 + y_t^2 - 2y_t y + y^2) = 0.$$

Now we separate the terms involving x and y:

$$\begin{aligned}(s_t^2 - s_p^2)x^2 + (s_t^2 - s_p^2)y^2 - 2(s_t^2 x_p - s_p^2 x_t)x - 2(s_t^2 y_p - s_p^2 y_t)y \quad &+ \\ s_t^2(x_p^2 + y_p^2) - s_p^2(y_t^2 + x_t^2) \quad &= \quad 0\end{aligned}$$

This is the equation of a conic section. If the speeds are different we can rearrange and complete the squares:

$$\left(x - \frac{s_t^2 x_p - s_p^2 x_t}{s_t^2 - s_p^2}\right)^2 + \left(y - \frac{s_t^2 y_p - s_p^2 y_t}{s_t^2 - s_p^2}\right)^2 =$$

$$\left(\frac{s_t^2 x_p - s_p^2 x_t}{s_t^2 - s_p^2}\right)^2 + \left(\frac{s_t^2 y_p - s_p^2 y_t}{s_t^2 - s_p^2}\right)^2 - \frac{s_t^2(x_p^2 + y_p^2) - s_p^2(y_t^2 + x_t^2)}{s_t^2 - s_p^2}$$

which is the equation of a circle with center

$$(c_x, c_y) = \left(\frac{s_t^2 x_p - s_p^2 x_t}{s_t^2 - s_p^2}, \frac{s_t^2 y_p - s_p^2 y_t}{s_t^2 - s_p^2}\right) \tag{3.5}$$

and radius

$$r = \frac{s_p s_t}{|s_p^2 - s_t^2|}\sqrt{(x_p - x_t)^2 + (y_p - y_t)^2} = \frac{s_p s_t}{|s_p^2 - s_t^2|}d. \tag{3.6}$$

If $s_p < s_t$, the cover set ($\mathcal{C}_{pt}$) is all points inside the circle, if $s_p > s_t$, $\mathcal{C}_{pt}$ is all the points outside the circle, if $s_p = s_t$, $\mathcal{C}_{pt}$ is limited by the perpendicular bisector of the line segment joining the agents, as shown before. When there are several pursuers and one target, the cover set is the union of the cover set for every pursuer: $\mathcal{C}_t = \bigcup_p \mathcal{C}_{pt}$. This is because as long as a point is covered by one pursuer the target cannot reach it. When there are several targets the cover set is the intersection of the cover sets for each target: $\mathcal{C} = \bigcap_t \left(\bigcup_p \mathcal{C}_{pt}\right)$. This is because as long as there is a target that can safely reach a point it is not covered.

As an illustration consider the case in which $s_t = 1$, $s_p = 0.5$, $(x_t, y_t) = (0, 0)$ and $(x_p, y_p) = (1, 0)$. The center of $\mathcal{C}$ is $(c_x, c_y) = (\frac{1}{1-0.25}, 0) \approx (1.333, 0)$ and the radius is $r = \frac{0.5}{|0.25-1|} \approx 0.666$. This is shown in Figure 3.3.

Unlike the case when $s_p = s_t$, when the speeds are different the cover-maximizing strategy may not be the best. When there is only one pursuer that is slower than the target, $|\mathcal{C}|$ decreases as they get closer; as $d \to 0$, $r \to 0$. Even then, $|\mathcal{C}|$ is a good heuristic: it minimizes the target's mobility by reducing the number of locations that the target can reach safely. Also, it automatically coordinates multiple pursuers because the cover set grows when the pursuers are separated; this leads to the pursuers spreading out and surrounding the target.

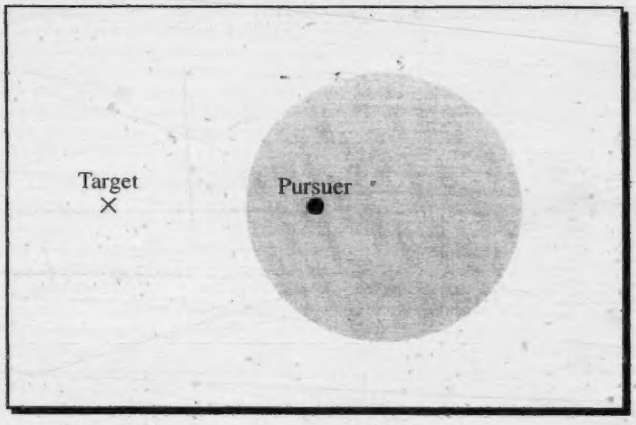


Figure 3.3: The cover set (gray) when the pursuer is slower than the target.

3.2.2 Graphs

We can apply the same ideas to graphs. The pursuer is at vertex $v_p \in V$ and the target is at $v_t \in V$. The pursuer can reach v in time $t(v_p, v) = c(v_p, v)/s_p$, where $c(v_1, v_2)$ is the cost of the shortest path between v_1 and v_2, similarly for the target. The set $\mathcal{C}$ consists of all vertices v for which $t(v_p, v) \leq t(v_t, v)$, given that the path from v_t to v does not include vertices in $\mathcal{C}$ and the path from v_p to v does not include vertices in $V \setminus \mathcal{C}$.

To build $\mathcal{C}$ it is necessary to start from the vertices that are nearby and do a breadth-first expansion so that the target paths do not intersect $\mathcal{C}$. Both the cover set and its complement, the target-cover set, are calculated at the same time. Vertices can be marked as belonging to the cover set, marked as belonging to the target-cover set, or unmarked. We start with all vertices unmarked and then mark all the vertices currently occupied by an agent. Then we proceed until there are no more unmarked reachable vertices (see Figure 3.4). It takes time linear in $(|E| + |V|) \log |V|$ to calculate the cover set, which is the main disadvantage of using it as a heuristic, but this can be mitigated by using state abstraction.

3.2.3 Advantages and disadvantages

To see why cover gives a better solution than geometric distance, lets consider the round-table map in Figure 3.1. As explained before, a geometric distance heuristic may lead two pursuers in the same direction, making them trail the target indefinitely. On the other hand, if each pursuer maximizes $|\mathcal{C}|$ independently, they will effectively separate and surround the target. This is because, if two pursuers stand in the same vertex, their cover is the same as if there were only one pursuer, but if they instead separate then they can increase the size of the cover set by covering separate regions. The fact that the cover set can be maximized independently is important because we can avoid exploring the joint-action space of all pursuers, which grows exponentially with the number of pursuers.

```
CalculateCover() : C
 1  q ← ∅
 2  for each agent a do
 3      add [vertex v(a), speed s(a), type T(a) ∈ {target, pursuer}, t = 0] to q
        (pop pursuers first if there is a tie)
 4  end for
 5  while q is not empty do
 6      pop the element [v, s, T, t] with smallest t from q
 7      if v is marked then continue
 8      if T = pursuer then
 9          mark v as covered
10      else
11          mark v as target-covered
12      end if
13      for each neighbour v' of v, connected with an edge of cost c do
14          if v' is not marked then
15              add [v', s, T, t + c/s] to q
16          end if
17      end for
18  end while
```

Figure 3.4: Cover set calculation algorithm.

In many situations, the cover heuristic gives a good estimate of the optimal solution (as will be shown in Section 4.1), but there are particular situations in which it is misleading, as can be expected with heuristic methods. One such situation is when a pursuer is slower than the target and the target and pursuer are close to each other. In this case the cover heuristic will tell the pursuer to stand back and "cover" some region instead of rushing in for the target which may be a better strategy against some kinds of targets. The cover heuristic is conservative in the sense that it will not lead the pursuer near the target unless there is a high chance of the target being captured. While this may be a good strategy against optimal or near-optimal targets, there are better strategies when the target is not so smart. We will deal with this problem in Section 3.3.

Another situation in which the cover heuristic may not give good results is when one pursuer is inside the area that another pursuer is already covering (see Figure 3.5). In this case all possible actions that the pursuer can take have the same value and therefore the pursuer has to make a uninformed decision. This leads to one pursuer using an effective strategy while the other moves randomly or stands still, which is definitely a bad idea. Fortunately, a simple tie-braking mechanism eliminates this problem, as will be shown in the next section.

Even after addressing these two problems, there are specific situations when the cover heuristic has non-global minima. Take for example the four-connected grid depicted in Figure 3.6: the target cannot move without being captured and the pursuers cannot move without giving the target a chance to move. By following the cover heuristic the pursuers will stay like that forever, even though they can capture the target in a few moves if they break the dead-lock. To avoid these minima the pursuers can look ahead, but this has the drawback of increasing the time complexity.

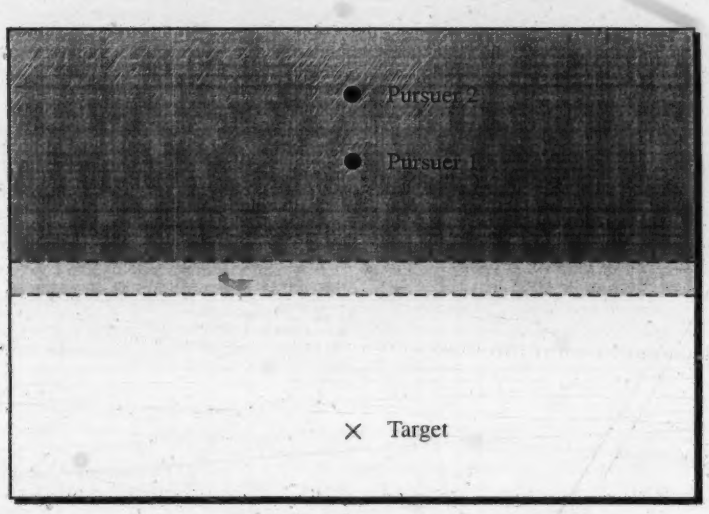


Figure 3.5: The cover set of two pursuers and one target with the same speed. The second pursuer's cover (dark gray) is inside the first pursuer's cover (light gray).

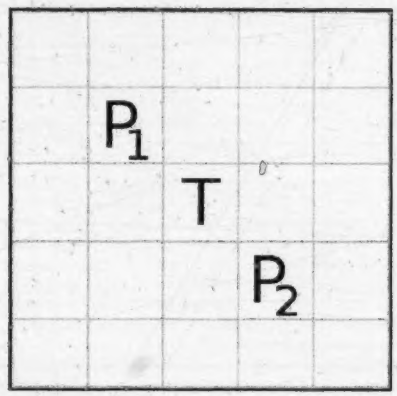


Figure 3.6: A local minimum of the cover heuristic. The target, T, can not move without being captured by the pursuers, P_1 and P_2.

3.3 CRA

A straight-forward way of using a heuristic function is with a greedy algorithm that, in each step, chooses the successor with the smallest heuristic value. For most heuristic functions this approach is not practical because the problem solver may get stuck in local minima. Fortunately this greedy approach gives acceptable results when using the cover heuristic — cover does have local minima but in many situations they are not detrimental to the overall result.

Although using a greedy algorithm by itself gives acceptable results, there are ways to improve the greedy approach. The previous section highlighted several drawbacks of the cover heuristic: it may be too conservative, it needs a good tie-braking mechanism, and it needs a way of avoiding local minima. Another drawback, mentioned earlier, is the time complexity of calculating the cover set.

Following are the proposed solutions to mitigate these drawbacks, resulting in the CRA algorithm.

3.3.1 Risk

The cover heuristic has a tendency towards being conservative as it may prefer vertices that cover a large region, over vertices in which the pursuer is close to capturing the target. One such situation is when the pursuer is slower than the target and there are no other pursuers close by to help surround the target. This "waiting" behavior can be advantageous here because a slow pursuer will wait for its teammates to surround the target, but there are situations where a more risky pursuer works better.

In the other extreme, a risky pursuer does not hold back at all — it only rushes toward the target, like a single-agent search algorithm guided by a geometric distance heuristic. While the risky strategy can be good against slow targets or targets that are not actively evading the pursuers, it does badly against fast or smart targets; the target waits for the pursuer to get near and then slips away.

We need an intermediate solution, where we can set the pursuers' riskiness depending on the type of target. To do this, two moves are generated: the one with the largest cover value $c_{\max}$, and one proposed by a single-agent search algorithm guided by a geometric distance heuristic, such as PRA*, with cover value c_r. The ratio of these two values, $c_r/c_{\max}$, determines if the single-agent-search move is actually risky or not.

We use a risk parameter, ρ, to choose between the move proposed by cover and the move proposed by, e.g., PRA*: if $\rho < 1 - c_r/c_{\max}$ then we choose the cover move, else we choose the PRA* move. A risk of zero means that cover is always used, a risk of one means that PRA* is always used, and a risk in $(0, 1)$ chooses PRA* only when it is not too risky. With this approach the "riskiness" of the algorithm can be controlled via ρ.

3.3.2 Abstraction

As mentioned before, the time it takes to compute the cover set is proportional to $(|E|+|V|)\log|V|$. If this is done on a graph with branching factor b and for each of the n pursuers then it takes time proportional to $bn|V|\log|V|$. This may not be feasible for real-time applications, as it has to be done at every time step. Abstraction can be used to mitigate this problem.

The abstraction methods were introduced in Section 1.4: an abstraction for the graph is built before the run begins, the CRA algorithm is run in the abstract graph, and the abstract actions are refined to obtain ground-level actions. By doing this, the complexity of calculating cover can be reduced to bn, with a high enough level of abstraction. The problem is that, as the abstraction level increases, the quality of the solution decreases. We therefore want to use the lowest abstraction level possible, given the time constraints. Also, abstraction introduces the time complexity required for the refinement process: $k\ell$ where k is the average abstract state size and ℓ is the level of abstraction. Therefore, the overall time complexity per turn is $O(bn|V|\log|V|k^{-\ell} + k\ell)$.

3.3.3 Tie-breaking

In some situations the cover heuristic can not differentiate between several moves because they all have the same value. A good tie-breaking mechanism is needed to avoid making uninformed moves. The PRA* move calculated when dealing with risk (Section 3.3.1) can be used for this: if there are several moves with a cover value of $c_{\max}$, and the PRA* move is one of them, then CRA selects the PRA* move as it moves the pursuer closest to the target.

Sometimes the PRA* move is too risky and it can not be used for tie-breaking because the decision is between conservative moves. In these situations we need another tie-breaking method. Here we want to avoid actions that lead to a vertex that is occupied by another pursuer and prefer actions that minimize the geometric distance. The actions that lead to a vertex that is occupied by another pursuer are removed (unless all actions lead to an occupied vertex) and from the remaining actions we select the one that minimizes the geometric distance heuristic (e.g., Manhattan distance). These tie-braking mechanisms boost the performance of CRA (see Chapter 4). The reason for this performance boost is that, when there are multiple agents, we often find that several moves are equivalent. If there is no tie-braking mechanism there will be many uninformed moves, which means that some of the pursuers will act randomly or stand still, which is a very bad strategy.

3.3.4 Avoiding local minima

The final concern is that the greedy method can lead to local minima, which tells the pursuers to stand still — all actions except *stand-still* lead to a higher heuristic value (see Figure 3.6). To avoid these situations, CRA forces the pursuers to move when no pursuer moved for a complete round. In most situations this will break the dead-lock.

Avoiding dead-locks is very important because, once in a dead-lock, no agent will move (unless the target makes a mistake). If no agent moves then the timer will expire and the pursuers will not get a chance to capture the target — the pursuers have given up. This is why it is important to avoid these minima, even if the target temporarily gains an advantage.

3.3.5 Putting it together: CRA

The Cover with Risk and Abstraction (CRA) algorithm that uses the mechanisms outlined in the previous sections is described in Figure 3.7. CRA receives three parameters: the pursuer p, the risk ρ, and the abstraction level ℓ, and returns an action for p. It first reduces the set of successor vertices of maximum cover with the tie-braking mechanisms (Lines 2 to 4, 9 to 12, and 13) and then compares the risk of the move proposed by PRA* with the allowed risk (Line 15) to choose the action and refine it.

We run this for each pursuer independently. This means that the time complexity of the algorithm is linear in the number of pursuers, as there is no need to analyze the joint-actions of all the pursuers. The time complexity of the cover calculation is $O(bn|V| \log |V| k^{-\ell} + k\ell)$, as shown before. To this

```
CRA(p, ρ, ℓ) : action
 1  A ← set of available abstract actions for p
 2  if no pursuer moved in the last turn then
 3      A ← A \ {stand-still}
 4  end if
 5  V_max ← the abstract successor vertices in A with maximum cover
 6  v_PRA* ← the abstract successor given by PRA*
 7  c_max ← the cover value of v ∈ V_max
 8  c_PRA* ← the cover value of v_PRA*
 9  remove from V_max vertices that are occupied by other pursuers
10  if V_max = ∅ then
11      restore V_max to its original contents
12  end if
13  v_max ← the vertex in V_max that minimizes the Manhattan distance
14  Δc = 1 − c_PRA*/c_max
15  if Δc > ρ then
16      return refine(v_max, ℓ)
17  else
18      return refine(v_PRA*, ℓ)
19  end if
```

Figure 3.7: The CRA algorithm.

we have to add the time complexity of calculating the PRA* move: $O(\sqrt{|V|}\log\sqrt{|V|}+k\log_k|V|)$. While CRA is not constant-time, if a high enough level of abstraction is used it can be remarkably fast, especially as it is considering the whole graph to make a decision.

If only local decisions were desired then a local cover function could be used that limits the cover calculation to a fixed depth and a local PRA* search could be performed as well. Limiting the size of this local space could produce a constant-time algorithm, however we leave this as a subject of future research.

3.4 Target's Cover

So far, we have explored the cover method from the *pursuers'* perspective; it can also be used as a target algorithm. For the targets, the algorithm consists of *minimizing* the cover, which corresponds to maximizing the target-cover. Most of the previous discussion applies for targets as well, except the tie-braking and the local minima. The tie-braking for a single agent is not a problem, as it is only a problem when dealing with multiple agents. This study considers only a single target, therefore the tie-braking issue is not a problem. The local minima is not a problem either because the target needs to escape from capture as long as possible, and standing still in a dead-lock is one way of achieving that. For these reasons the target's cover algorithm consists simply of a greedy algorithm with abstraction.

3.5 Theoretical properties of Cover

Conjecture 3.5.1. *For the uniform speed problems, the cover heuristic gives exact solution costs in graphs where one pursuer has a winning strategy against one target.*

Proof outline. According to Aigner and Fromme [2, Theorem 1], one pursuer is sufficient iff the graph can be reduced to a single vertex v_0, by successively removing pitfalls. A pitfall is a vertex p such that there is a dominating vertex d, that is, $N[p] \subseteq N[d]$ (i.e., all neighbors of p are also neighbors of d), as shown in Figure 3.8. Suppose that for graph $G_0 = (V, E)$ one pursuer is sufficient. To reduce G_0 we remove all pitfalls $p \in P_0$ dominated by a vertex $d \in D_0$ to get $G_1 = (V \setminus P_0, E \setminus \mathit{edges}(P_0))$, then remove P_1 to get G_2 and so on until $D_n = \{v_0\}$ and $P_n = \emptyset$ on $G_n = (\{v_0\}, \emptyset)$.

If there is a strategy for G_i for $0 < i \leq n$ then a related strategy can be applied to G_{i-1}: for all vertices in G_i the strategy is the same, for vertices in P_{i-1} the pursuer moves to a vertex not in P_{i-1} and then follows the same strategy. If the target steps into any vertex in P_{i-1} the pursuer assumes that the target is in the dominating vertex in D_{i-1} (which is part of G_i). This strategy is guaranteed to be as effective as in G_i and to take at most two steps more: one if the pursuer has to step out of P_{i-1} and another when going from the dominating vertex to capture the target.

Starting from G_n the strategy can be expanded to G_0: go to v_0 and then follow the vertices that dominate the current location of the target, unless the starting vertex already dominates the subgraph where the target is, in which case there is no need to go to v_0, just to follow the vertices that dominate the current location of the target. This is the optimal strategy against an optimal target because when the pursuer is at a dominating vertex the target can not escape from that subgraph while stepping into a vertex that does not dominate the target's vertex gives the target more room to move and therefore wastes time. To show that a pursuer guided by the cover heuristic follows this strategy note that v_0 maximizes the cover set over all possible target vertices. If the pursuer is in a vertex that leads to vertices that dominate the target's vertex then the pursuer will move so as to increment the number of vertices it dominates, which increases the cover set. If the target is in a vertex dominated by the pursuer's vertex then the cover will be maximized by capturing the target. □

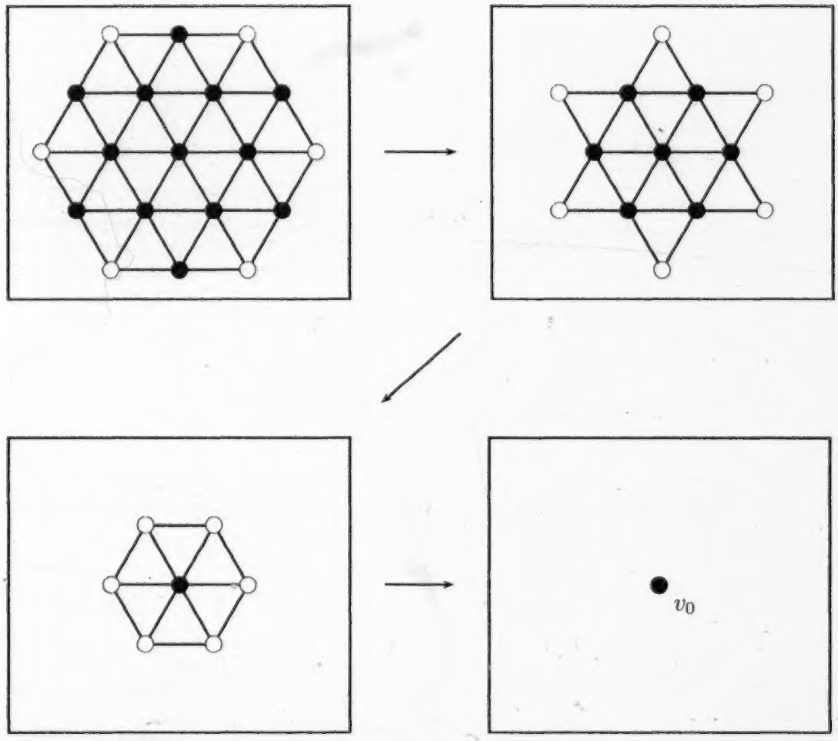


Figure 3.8: A graph in which one pursuer can always capture one target. By removing pitfalls (empty circles) it can be reduced to a single vertex, v_0.

Chapter 4

Empirical Evaluation

To evaluate the performance of the methods in a practical context, we use a framework for testing different pursuit algorithms against different target algorithms. The framework, as outlined on Section 2.2, supports any number of agents with non-uniform speeds. Given a map, the framework generates random starting positions and lets the algorithms run until the target is captured or until the timer expires. We use timeouts to avoid infinite runs, using a time for each simulation that depends on the size of the map and is set at least one order of magnitude larger than the average capture time.

Several different pursuit algorithms are used to be able to compare a range of behaviors in each situation, Table 4.1 gives the details. Likewise, a number of different target algorithms are used: Table 4.2. Depending on the study, different combinations of these targets are tested against pursuers. The optimal solver can only be used for small maps with a small number of agents due to its time complexity. Also, the algorithms based on minimax are too slow when there is a large number of agents so they are used only in the smaller problems.

Abbreviation	Description	Further details
OPT	Optimal solver.	Section 2.1
CRA(ρ, ℓ)	Cover with Risk and Abstraction using risk ρ and abstraction level ℓ.	Section 3.3
MTS	The improved version of MTS with $C = D = 10$.	Section 1.3
PRA*	Partial Refinement A*.	Section 1.4
MMX(d)	Minimax to a depth of d plies using geometric distance as the evaluation function.	[32]
MMC(d)	Minimax to a depth of d plies using cover as the evaluation function.	[32]

Table 4.1: The pursuit algorithms.

Abbreviation	Description	Further details
OPT	Optimal solver.	Section 2.1
TGC	Greedy target guided by the cover heuristic.	Section 3.4
SFL	Simple Flee target randomly places beacons on the map and uses PRA* to get a path to the beacon that is furthest from the pursuers. It follows that path for five moves and then chooses a beacon again.	–
DAM(d)	Dynamic Abstract Minimax runs minimax at the highest level of abstraction. If an escape from the pursuers is not found then it goes to a lower level of abstraction and the process repeats. When an abstract action that leads to escape is found it is used to guide the target.	[8]
MMX(d)	Minimax to a depth of d plies using geometric distance as the evaluation function.	[32]
MMC(d)	Minimax to a depth of d plies using cover as the evaluation function.	[32]

Table 4.2: The target algorithms.

4.1 Optimal solution

The first evaluation consists of comparing the proposed approach to the optimal strategy. The optimal strategy can only be computed for small maps with few agents, therefore in this part five maps are used: an empty 5×5 rectangular grid, a 9×9 grid with a round table in the middle, a 10×11 maze grid, a dodecahedron on which three pursuers are necessary (Figure 1.1) and an hexagonal grid on which one pursuer is sufficient (Figure 3.8); the maps are illustrated in Figure 4.1. Notice that this is only a small sample of all possible graphs in which cover may be used, it is the subject of future research to determine if these results apply in the general case.

When comparing we will use four types of targets — OPT, TGC, DAM, and SFL — and four types of pursuers — OPT, CRA($\rho = 0, \ell = 0$), MTS, and PRA*. The number of pursuers used for each map is the smallest required for capture, therefore on the grid maps the number of pursuers is 2, on the dodecahedron the number is 3, and on the hexagonal grid the number is 1. All agents have the same speed, each goes from one vertex to the next in exactly one time step.

The principal performance measure used is success rate — the percentage of runs in which the pursuers capture the target. There is one run for every possible joint state on each map. The optimal pursuer always has a success rate of 100%, as expected (Figure 4.1), and the optimal target has the highest escape rate (lowest success rate for the pursuers), in most cases. It can happen that a non-optimal target has a higher escape rate than the optimal target with a sub-optimal pursuer (as happens on the maze map for CRA chasing TGC) because the optimal target is optimized for the

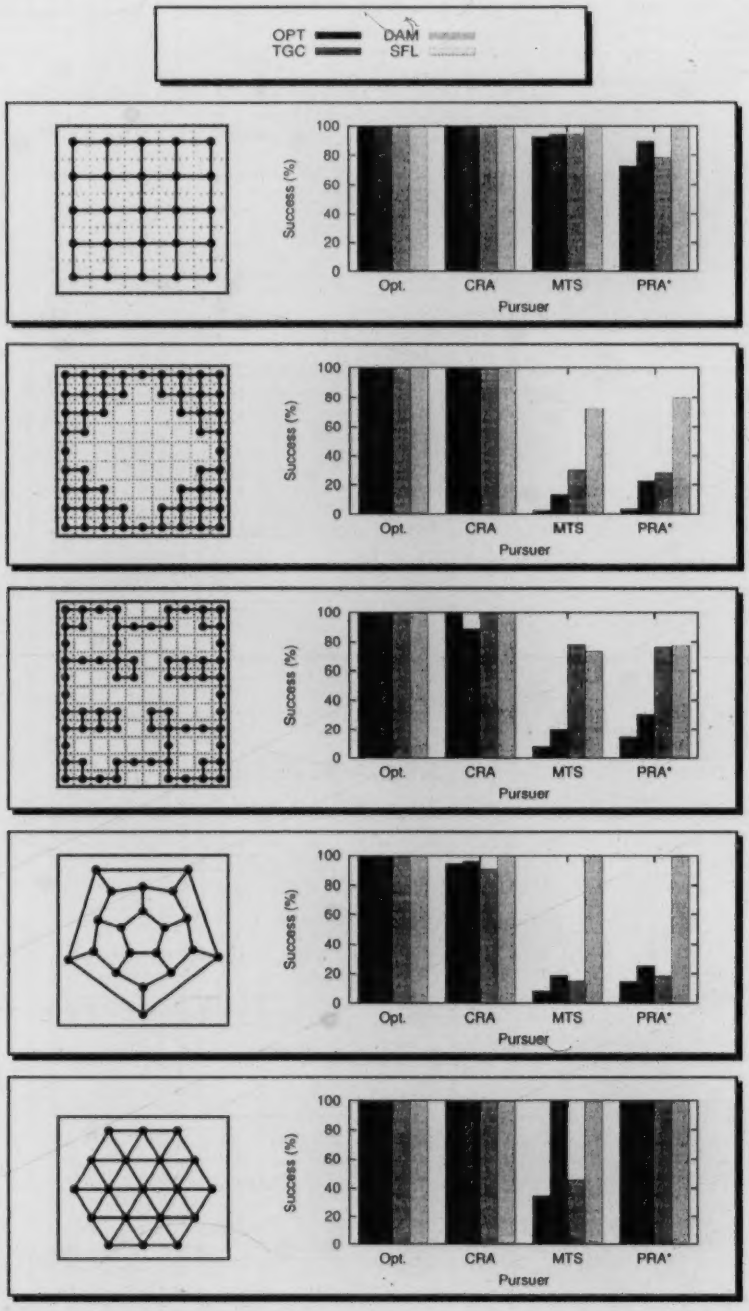


Figure 4.1: The success rate on five different maps of several pursuer algorithms after several target algorithms.

optimal pursuer; if the pursuer is sub-optimal the optimal target is not necessarily the best target algorithm.

Figure 4.1 shows that CRA has a success rate rate of 100% on all maps except the maze and the dodecahedron on which the success rate is slightly smaller against some targets. Compare this to MTS whose success rate is as low as 2.6% on the round-table map when chasing an optimal target. MTS is successful when the distance heuristic gives good results, like on an empty map, and when no coordination is needed, like on the hexagonal grid where there is only one pursuer. Even on these situations the performance of MTS is below CRA. This is true for PRA* as well, but notice that PRA* does better than MTS when no coordination is necessary. These results highlight the importance of having a heuristic that can help coordinate multiple agents.

Also, it is important to note that the performance of the target that uses cover is close to the performance of the optimal target. By contrast, Simple Flee has very low escape rate (very high success rate for the pursuers). For instance, Simple Flee never escapes in the dodecahedron map. These results are another manifestation of the adequacy of the cover heuristic for both pursuers and targets.

While the most important performance measure is success rate, it is also useful to measure time to capture. The capture time is considered to be infinite for problems on which the pursuers did not capture the target, we therefore take the average only for the problems where the pursuers succeeded. Unfortunately, this filtered measure biases the result because low success algorithms are more likely to solve the easy problems than the hard ones. For this reason, to have a meaningful comparison, we also compute the average capture time for the optimal pursuer *in the subset of problems that a given pursuit algorithm solved.* Table 4.3 shows the optimal capture time for the subset of problems solved by each algorithm and the capture time for the algorithm over that same subset when chasing an optimal target.

We see that low success algorithms are solving easy problems because the average optimal capture time for that subset of problems is smaller than the average optimal capture time for all the problems. This is especially true for the round table map: here the optimal capture time on the problems that MTS solved is half of the optimal capture time for all problems. The hexagonal grid is interesting because all optimal capture times are nearly the same. While CRA is optimal (see Section 3.5), the other algorithms also have a very high success rate. On this map the capture time for the MTS algorithm is more than double the optimal, showing that MTS can take a long time to capture even when there is no coordination involved.

To have more reference points, lets compare the minimax algorithm guided by both distance (MMX) and cover (MMC) heuristics for both pursuers and targets. For the pursuers MMC is not as effective as CRA, using depths up to 5 plies. Looking back at Section 3.2, this can be readily explained: the cover heuristic, by its own, has some drawbacks. But even then, MMC is much better than MMX, for the same depth. For instance, when running on the maze map against an optimal

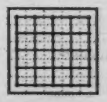
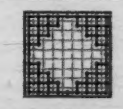
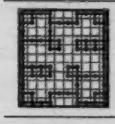
Map	Algorithm	Success (%)	Opt. time	Time
	OPT	100.0	4.993	4.993
	CRA	100.0	4.993	5.670
	PRA*	72.8	4.896	13.22
	MTS	92.4	4.963	14.69
	OPT	100.0	15.52	15.52
	CRA	100.0	15.52	17.89
	PRA*	3.7	7.757	8.775
	MTS	2.6	5.155	6.190
	OPT	100.0	19.14	19.14
	CRA	100.0	19.14	21.27
	PRA*	14.8	12.68	24.81
	MTS	8.2	9.646	12.20
	OPT	100.0	3.309	3.309
	CRA	94.6	3.300	4.351
	PRA*	14.6	2.652	3.777
	MTS	8.4	2.273	6.533
	OPT	100.0	4.351	4.351
	CRA	100.0	4.351	4.351
	PRA*	99.4	4.347	10.60
	MTS	34.5	4.331	7.441

Table 4.3: The optimal time for the subset of the problems solved by each algorithm and the time the algorithm took to solve them.

target MMC(3) has a success rate of 42.8% while MMX(3) has a success rate of only 5.5%. This is because of the advantages of using cover instead of distance, as explained in Chapter 3.

When using MMC($d > 1$) as a target algorithm we find that it is more effective than TGC. This is easily explained as TGC is basically a depth 1 minimax search. A target using MMC is still much more efficient than one using MMX, for the same reasons as in the pursuit case. While it could be advantageous to use minimax with cover heuristic as a target algorithm, the gains in performance do not justify the increment in run time. For instance MMC(5) has an escape success rate against MTS of 82.2% which is comparable to the 80.0% success rate for TGC.

This demonstrates that the plain minimax algorithm is not a good contestant when using small depth values. While this algorithm is better for larger depth values, it then becomes so slow as to be impractical. For these reasons we do not use these algorithms in the remainder of the empirical evaluation.

4.2 Coordination

To explore the coordination capabilities of CRA, we choose a map on which both MTS and PRA* fare well: an empty 20 × 20 grid. Notice that, in this map, 20 pursuers could sweep the map and always capture the target, regardless of their speed. The target algorithm is fixed to TGC. The speed of the target is fixed to 1.0 while the speed of the pursuers varies from 0.2 to 1.0 in increments of 0.2. One hundred runs are executed with random starting locations for all agents, using the same starting

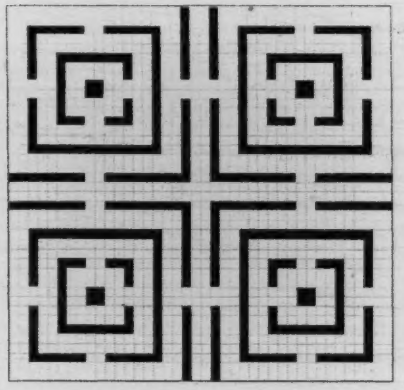


Figure 4.2: A hand-made map with obstacles

locations across problem instances. The extended Wald method is used to determine the confidence intervals of the results, with 95% confidence [1]. This test gives the binomial proportion confidence interval and is more accurate that the "exact" method by Clopper and Pearson [12]. Figure 4.3 shows the results. As the number of pursuers increases, the success rate of all the algorithms increases. CRA has the most pronounced increase, being able to capture the target more often than not with 50 pursuers having a speed that is one fifth of the target's speed, while MTS never achieves a success rate of even 5% with such speed. This result is very statistically significant. Here PRA* does considerably better than MTS, especially with low speeds.

We use a more involved map to see the advantages of coordinated cooperation when there are obstacles that can be used to help surround the target: the map in Figure 4.2. The conditions are the same: 100 runs in random starting locations with a TGC target. The results are shown in Figure 4.4. This problem is evidently more difficult because the target can run around obstacles to avoid capture. Here the difference between CRA and the other pursuit algorithms is more pronounced because the target has more chances of running in circles if the pursuers are not coordinated. We again see the advantages of having coordinated agents.

As the speed decreases the number of pursuers necessary for capture increases super-linearly. The exact relation between speed and number of pursuers is hard to establish as it depends on the map and, because with more than a few pursuers it is not viable to calculate the optimal strategy, we cannot compute the optimal solution. Nonetheless, given that both CRA and TGC are good approximations of the optimal strategies, it can be conjectured that the increase shown is close to the real relation.

4.3 Abstraction

Until now, the experiments focused on the effectiveness of the algorithms, without regard to the computation time needed to achieve those results. We now compare the computation times. It is

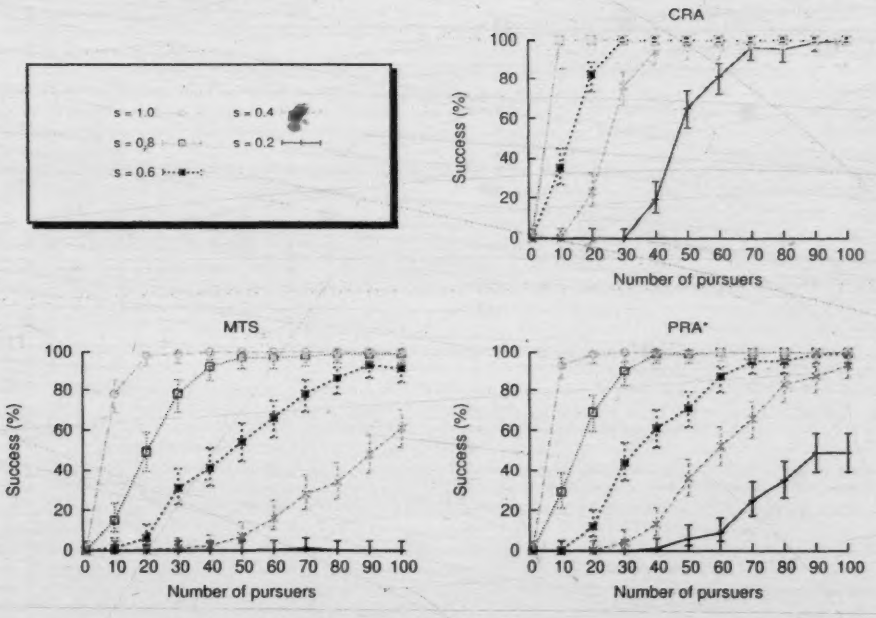


Figure 4.3: The success rate as a function of the number of pursuers on an empty 20×20 grid for different speeds (s). The pursuer algorithms are CRA (top right), MTS (bottom left), and PRA* (bottom right). The target algorithm is TGC in all cases. The confidence intervals for 95% confidence are shown.

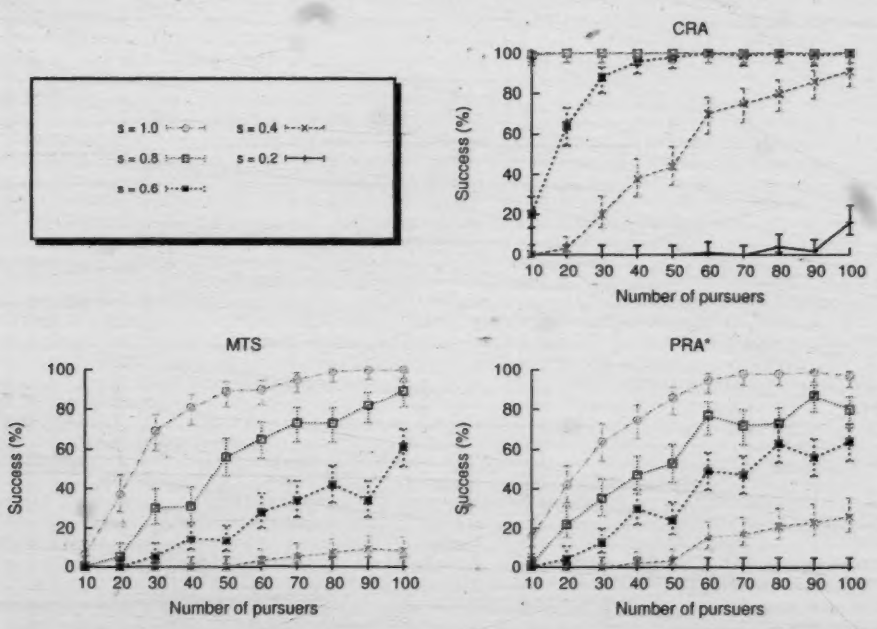


Figure 4.4: The success rate as a function of the number of pursuers on a grid-world with obstacles for different speeds (s). The pursuer algorithms are CRA (top right), MTS (bottom left), and PRA* (bottom right). The target algorithm is TGC in all cases. The confidence intervals for 95% confidence are shown.

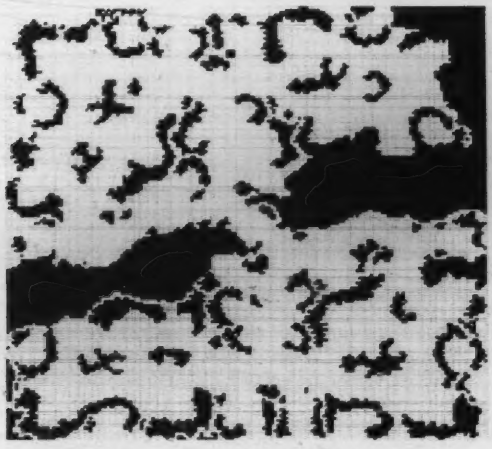


Figure 4.5: A map from a commercial video game.

clear that CRA with no abstraction is too slow on big maps, see Section 3.3. For this reason we will now consider abstraction and compare the computation time to the degradation in solution quality.

This is done using a commercial-game map with nearly twenty thousand states, as shown in Figure 4.5. Unlike to the grid maps considered so far, this map is eight-connected and therefore the number of pursuers necessary is larger. For this particular map three pursuers are necessary and sufficient to capture the target. The results are presented for both two and three pursuers.

First, we compare the run time per move, per agent for the pursuit algorithms. Figure 4.6 shows that, with at level of abstraction of 5, the run time per move, per agent of CRA is comparable to that of PRA*, while MTS is more than one order of magnitude faster. The fact that MTS is faster is expected because MTS is a constant-time algorithm, while PRA* and CRA are not. Nonetheless, CRA has a reasonable run time per move, of only 0.64 milliseconds, which is a small price to pay for the improved success rate. Note that the run time per move, per agent remains constant regardless of the target algorithm or the number of pursuers as it depends only on the size of the map. The variance for the results shown in Figure 4.6 is too small to be noticeable in the graph — the results have little noise.

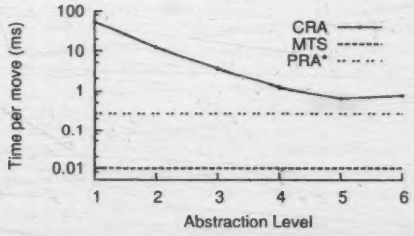


Figure 4.6: The run time per move, per agent for the pursuit algorithms. For CRA the time is shown for different levels of abstraction.

Now we consider the decrease in success rate. Depending on the situation, there may be a decrease in success rate when the level of abstraction is increased, as can be seen in Figure 4.7. Here two pursuers are chasing one target and the pursuers and target have the same speed. For this map, two pursuers cannot capture an optimal target; the pursuers have to rely on the target making mistakes. For DAM and SFL targets the decrease in success rate for CRA is small and goes below the success rate of MTS only for abstraction levels 4 or above. TGC, being a much better target algorithm, manages to escape most of the time, in this case CRA stays above the other pursuers for all levels of abstraction. The extended Wald method is used to determine the confidence intervals, as shown. For SFL the results are not statistically significant and only the confidence interval for CRA is shown.

When there are three pursuers there is a winning strategy. Figure 4.8 shows that in this case the decrease in success rate due to abst[illegible]s much more pronounced when chasing TGC. Nonetheless, the success rate of CRA stays above the success rate of both MTS and PRA* for all levels of abstraction below 6 when chasing TGC and DAM. When chasing SFL the success rate of CRA is the same as MTS above level 4. Again, the results fro SFL are not statistically significant and only the confidence interval for CRA is shown.

We see that abstraction does affect the solution quality, by an amount that depends on the situation. The worst case for this map (Figure 4.5) happens when three pursuers are chasing TGC, but the success rate of MTS and PRA* is so low that, even with the big penalty of abstraction, CRA comes out on top. There is a trade-off between how long CRA takes to run and how well it does. Therefore, for each particular application, we need to empirically determine the appropriate abstraction level.

4.4 Risk

The success rate of CRA can be increased even further in some situations with the use of the risk parameter, ρ. For example, in the map shown in Figure 4.5 with two pursuers chasing a TGC target, the success rate with $\rho = 0$ is higher than MTS and PRA* (Figure 4.7). It is even higher with $\rho = 0.1$, as shown in Figure 4.9. However, increasing risk is not always beneficial, for instance when chasing SFL (Figure 4.9).

The specific situations in which increasing risk may help have to be empirically determined. Nonetheless, from the experiments conducted we find that risk helps in situations in which the pursuers do not have a winning strategy, for example in the situation just mentioned. The fewer chances the pursuers have of capturing the target the more gains in taking risks. For instance, SFL is an easy target and therefore being risky may not be advantageous. On the other hand, TGC is a hard target and therefore we need to take risks to increase the chance of success.

This can be explained looking back at Section 3.3.1: the cover heuristic is conservative. If the pursuers realize that they do not have a clear winning strategy they stand back and cover a large region instead of rushing in. Therefore, taking risks is better when the pursuers have a disadvantage.

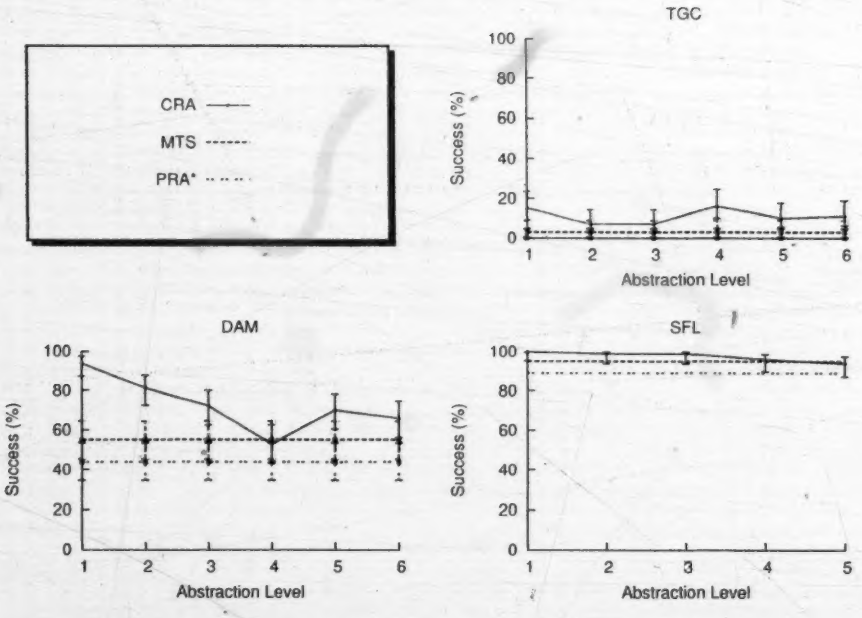


Figure 4.7: The success rate of the pursuit algorithms using two pursuers and chasing TGC (top right), DAM (bottom left) and SFL (bottom right). For CRA the success rate is shown for different levels of abstraction.

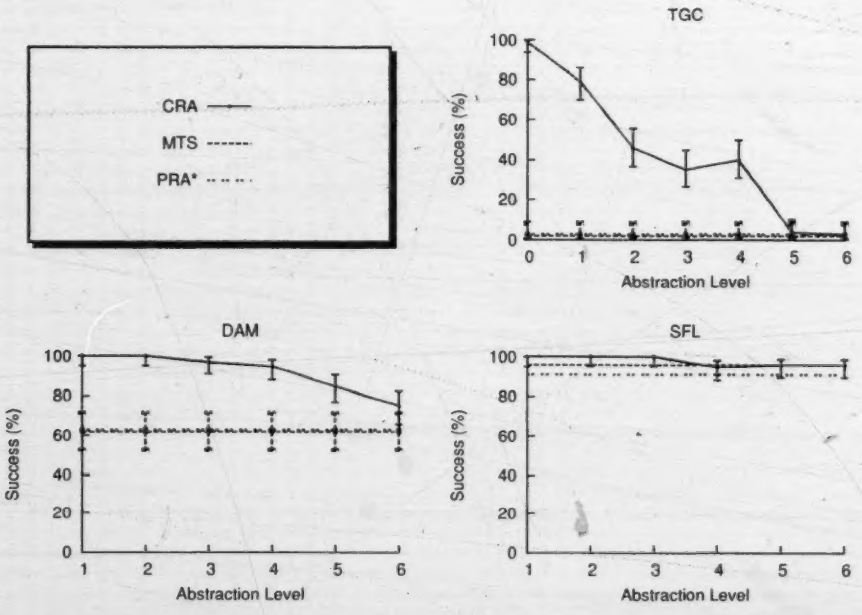


Figure 4.8: The success rate of the pursuit algorithms using three pursuers and chasing TGC (top right), DAM (bottom left) and SFL (bottom right). For CRA the success rate is shown for different levels of abstraction.

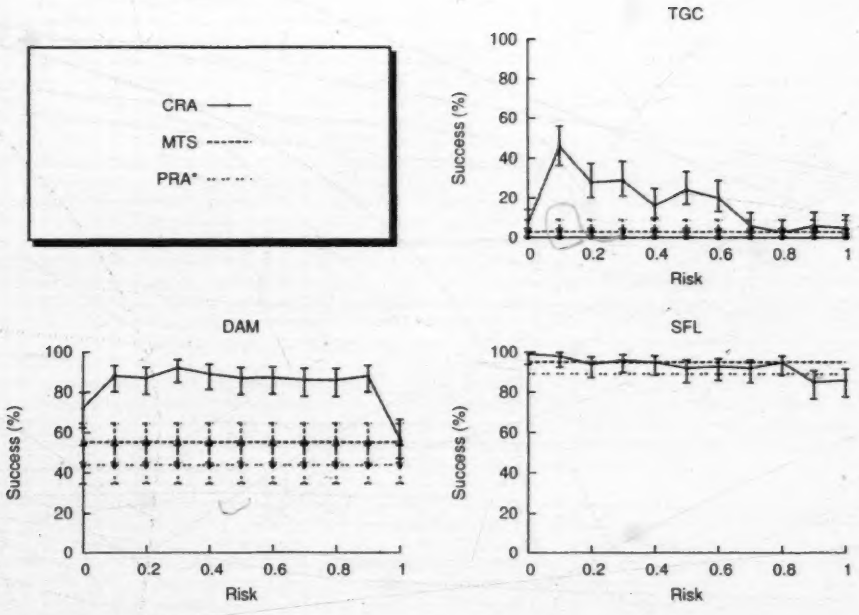


Figure 4.9: The success rate of the pursuit algorithms when chasing TGC (top right), DAM (bottom left) and SFL (bottom right). For CRA the success rate is shown for different values of risk.

Conclusions

Single-agent and multi-agent moving-target search problems are common in real-life situations. They are interesting problems because of their complexity and their applicability in military simulations, video games and law enforcement, to name a few. Previous approaches to the moving-target search problem treat it as a single-agent search problem — they use single-agent search algorithms with single-agent search heuristics. Because of this, previous approaches do not try to minimize the target's mobility or to coordinate multiple pursuers and therefore they have a low success rate.

This dissertation demonstrated that, on the maps used, an effective and efficient solution to the multi-agent moving-target search problem can be achieved using the information given by the cover heuristic. The cover heuristic is used in the CRA and TGC algorithms, which improve the state of the art for pursuers and targets, respectively. Some theoretical results were given about the algorithms and in general about the moving-target problem.

The theoretical results developed here (Section 2.1) showed that two pursuers are necessary and sufficient in an empty grid and that, in general, two pursuers are not sufficient in grid-worlds. With a new variable-speed framework we found that the theory from the uniform speed problem does not generalize to the variable-speed problem. This gives insight into the complexity of having arbitrary speeds, which makes the problem more interesting.

We explored the need to have a better heuristic than geometric distance by showing that the geometric distance heuristic is unsuitable for the moving-target problem, which explains the main drawback of previous approaches. We then introduced the cover heuristic motivating it and showed that it gives optimal results in some cases: in graphs for which one pursuer has a winning strategy and in Euclidean space against a target of the same speed. Cover is straight-forward to calculate both in arbitrary graphs and in Euclidean space. Based on this, we developed a new algorithm, CRA, and incorporated methods to address each of the drawbacks of using only cover. Experimental results showed that CRA is within 10% of optimal in all the problems for which the optimal strategy was calculated. Likewise, when compared to the optimal target, TGC has a very high escape rate. The results show how CRA can effectively use multiple slow pursuers to capture a fast target, in contrast to previous approaches that need many more pursuers to achieve a reasonable success rate. This demonstrates how the pursuers guided by cover can exploit the target's weaknesses. We also showed that sometimes taking risks can be rewarding, especially when the pursuers have a disadvantage.

These experiments were run on hand-made and commercial-game maps, which is a subset of graphs that have some real-life applications, it remains to be seen if these results hold for arbitrary graphs.

Overall, the results presented highlight the importance of, first, reducing the mobility of the target and, second, coordinating multiple pursuers. Likewise, they shed light on the target's strategy: preserve mobility instead of simply running away and perhaps cornering itself. The round-table example illustrates this (the map is shown in Figure 3.1 and the example is developed throughout Chapter 3).

The biggest drawback of the cover-based approach is its run time. Although the run time per move can be below one millisecond for a relatively large map, CRA is not constant-time, it depends on the size of the graph. Abstraction tries to mitigate this problem but a constant-time solution would be desirable; it is important in real-time situations like video games. Even with this drawback, the cover-based algorithm improves in success rate and capture time compared with previous state-of-the-art algorithms, showing that the cover heuristic is an effective solution to moving-target search problems.

Future work

The work presented here can be extended in several interesting directions. For instance, it can be extended to work in Euclidean space in both two and three dimensions. Section 3.2 presents the base cover calculation for a plane, it needs to be extended to be able to deal with obstacles, if required, so that it can be used with a continuous planning algorithm. This has applications in robotics and can lead to interesting coordinated behavior.

Another extension is to find other ways to speed up the cover computation without much loss in success rate. We analyzed abstraction methods but they have drawbacks because the abstractions are not explicitly designed for cover computations. If the abstraction is specifically tailored for cover then perhaps the cover computation can be sped up. Furthermore, it would be ideal to have a constant-time algorithm based on cover, perhaps by calculating cover only on a subset of states.

A disadvantage of CRA is that it has the risk parameter, ρ, whose best value may be hard to determine. By incorporating a dynamic risk policy, the algorithm could automatically determine the risk value. Section 4.4 gives an outline of the situations in which an increase in risk is advantageous but further study of this subject is necessary. An idea that was investigated briefly is using weighted cover — instead of using the number of covered states as the heuristic estimate we can give more weight to vertices that are close to the target. Preliminary results show that this can increase the success rate of CRA even further.

Another point worth considering is using TGC with abstraction. This idea was mentioned in Section 3.4 but the preliminary empirical studies determined that its performance decreases too much because of the refinement process. An adequate refinement process for targets needs to be determined and applied to TGC.

This thesis establishes a baseline for a new generation of pursuit and evasion algorithms. Hopefully this will serve as a motivation for the development of improved methods.

Bibliography

[1] Alan Agresti and Brent A. Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52:119–126, 1998.

[2] Martin Aigner and Michael Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1):1–12, 1984.

[3] Thomas Andreae. Note on a pursuit game played on graphs. *Discrete Applied Mathematics*, 9(2):111–115, 1984.

[4] Thomas Andreae. On a pursuit game played on graphs for which a minor is excluded. *Journal of Combinatorial Theory, Series B*, 41(1):37–47, 1986.

[5] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *DEDS*, 13:341–379.

[6] Anthony Bonato, Peter Golovach, Gena Hahn, and Jan Kratochvil. The search-time of a graph. *Discrete Mathematics*, 2008.

[7] Adi Botea, Martin Müller, and Jonathan Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1(1):7–28, 2004.

[8] Vadim Bulitko and Nathan Sturtevant. State abstraction for real-time moving target pursuit: A pilot study. In *Proceedings of the National Conference on Artificial Intelligence (AAAI), Workshop on Learning for Search*, pages 72–79, 2006.

[9] Vadim Bulitko, Nathan Sturtevant, and Maryia Kazakevich. Speeding up learning in real-time search via automatic state abstraction. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1349–1354, 2005.

[10] Vadim Bulitko, Nathan Sturtevant, Jieshan Lu, and Timothy Yau. Graph abstraction in real-time heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, 30:51–100, 2007.

[11] Fumihiko Chimura and Mario Tokoro. The trailblazer search: a new method for searching and capturing moving targets. In *Proceedings of the National Conference on Artificial intelligence (AAAI)*, pages 1347–1352, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[12] C.J. Clopper and E.S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26:404–413, 1934.

[13] Mark Goldenberg, Alex Kovarsky, Xiaomeng Wu, and Jonathan Schaeffer. Multiple agents moving target search. *International Joint Conference on Artificial Intelligence (IJCAI) - Poster*, pages 1538–1538, 2003.

[14] Arthur S. Goldstein and Edward M. Reingold. The complexity of pursuit on a graph. *Theoretical Computer Science*, 143(1):93–112, 1995.

[15] Russell Greiner. personal communication, 2008.

[16] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[17] Robert C. Holte, T. Mkadmi, Robert M. Zimmer, and Alan J. MacDonald. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence*, 85(1–2):321–361, 1996.

[18] Robert C. Holte, M. B. Perez, Robert M. Zimmer, and Alan J. MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*, pages 530–535, 1996.

[19] Alejandro Isaza, Jieshan Lu, Vadim Bulitko, and Russell Greiner. A cover-based approach to multi-agent moving target pursuit. In *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2008.

[20] Alejandro Isaza, Csaba Szepesvári, Vadim Bulitko, and Russell Greiner. Speeding up planning in Markov decision processes via automatically constructed abstractions. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008.

[21] Toru Ishida. *Real-Time Search for Learning Autonomous Agents*. Kluwer Academic Publishers, 1997.

[22] Toru Ishida and Richard E. Korf. Moving target search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 204–211, 1991.

[23] Toru Ishida and Richard E. Korf. Moving-target search: A real-time search for changing goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):609–619, 1995.

[24] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[25] Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

[26] Richard E. Korf and Larry A. Taylor. Finding optimal solutions to the twenty-four puzzle. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, pages 1202–1207, Portland, Oregon, USA, 1996. AAAI Press / The MIT Press.

[27] Nimrod Megiddo, S. Louis Hakimi, M. R. Garey, David S. Johnson, and Christos H. Papadimitriou. The complexity of searching a graph. 35(1):18–44, 1988.

[28] Nicolas Nisse and Karol Suchan. Fast robber in planar graphs. In *Proceedings of the 34th International Workshop on Graph-Theoretic Concepts in Computer Science*, 2008.

[29] Torrence D. Parsons. Pursuit-evasion in a graph. *Lecture Notes in Mathematics*, 642:426–441, 1978.

[30] Alain Quilliot. A short note about pursuit games played on a graph with a given genus. *Journal of combinatorial theory, Series B*, 38(1):89–92, 1985.

[31] John H. Reif. Universal games of incomplete information. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 288–308, 1979.

[32] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter 6, pages 163–171. Prentice Hall, Upper Saddle River, NJ, 2nd edition, 2003.

[33] Nathan Sturtevant and Michael Buro. Partial pathfinding using map abstraction and refinement. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1392–1397, 2005.